Jochen Ziegenbalg

Programmieren lernen mit Logo

Einführung in das interaktive Programmieren mit zahlreichen Lernbeispielen



Ziegenbalg · Programmieren lernen mit Logo

Jochen Ziegenbalg

Programmieren lernen mit Logo

Einführung in das interaktive Programmieren mit zahlreichen Lernbeispielen



Carl Hanser Verlag München Wien

Prof. Dr. rer. nat. Jochen Ziegenbalg ist Dozent an der Pädagogischen Hochschule Reutlingen und betreut die Arbeitsgebiete Mathematik und Informatik.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Ziegenbalg, Jochen:
Programmieren lernen mit Logo: Einf. in d. interaktive Programmieren mit zahlr. Lernbeispielen/Jochen Ziegenbalg. — München; Wien: Hanser, 1985. — ISBN 3-446-14441-2

Satz: Stummer, München
Druck und Bindung: Huber, Dießen

Printed in Germany

Vorwort

Warum denn schon wieder eine neue Programmiersprache? Diese Frage wird regelmäßig und zu Recht gestellt, wenn eine neue Sprache auftaucht. Jede neue Programmiersprache muß es sich gefallen lassen, mit bereits vorhandenen Sprachen verglichen zu werden. Hat sie im Vergleich zu diesen Sprachen keine Vorzüge aufzuweisen, so ist sie überflüssig.

Jede neue Sprache wäre überflüssig, wenn es unter den alten Sprachen eine "perfekte" gäbe. Dies ist nicht der Fall. Bezogen auf die Mikrocomputer-Szene (nur die steht hier zur Diskussion) läßt sich zur Zeit folgendes konstatieren: BASIC hat den Vorzug der Interaktivität; es weist aber im Bereich der Strukturierbarkeit unakzeptable Schwachstellen auf. Pascal ist strukturierbar, seine Interaktivität läßt aber sehr zu wünschen übrig. Was für Pascal gilt, dürfte in ähnlicher Weise auf für ELAN gelten. In Comal wurde die algorithmische Struktuirerbarkeit von BASIC erheblich verbessert, die Daten sind aber nach wie vor praktisch nicht strukturierbar. Forth ist in mancher Hinsicht faszinierend, seine Syntax ist aber sehr gewöhnungsbedürftig und seine Stukturierbarkeit ist nicht so ausgeprägt wie die von Pascal. Alle diese Sprachen sind darüberhinaus in der Symbolverarbeitung nicht besonders leistungsfähig. Keine dieser Sprachen weist einen strukturierten und zugleich dynamischen Datentyp auf, der sich mit der Liste, dem grundlegenden Datentyp von Logo, vergleichen könnte.

Logo ist nicht überflüssig. Es weist einen höheren Grad sowohl an Interaktivität als auch an Strukturierbarkeit auf, als alle oben genannten Sprachen. Es wird diese Sprachen dennoch nicht "schlagartig" ersetzen, denn in den zur Zeit bestehenden Versionen ist Logo als interpretierende Sprache bei vielen (nicht bei allen) Anwendungen ziemlich langsam; besonders dann, wenn man es mit compilierenden Sprachen (etwa mit Pascal) oder mit FORTH vergleicht. Dies kann sich ändern, wenn erst einmal Logo-Compiler zur Verfügung stehen. Die ersten davon tauchen gerade am Markt auf. Außerdem sind die gegenwärtigen Logo-Versionen meist etwas schmalbrüstig im Bereich der mathematischen Standardfunktionen (einschließlich der Feldverarbeitung) ausgestattet. Auch dieser Zustand ändert sich sehr schnell bei Mikrocomputern der 16- bzw. 32-Bit Generation, die durchweg einen sehr viel größeren Speicher haben als die Geräte der 8-Bit-Generation.

Wie Logo im Gesamtvergleich zu Prolog oder Smalltalk abschneiden wird, ist noch unklar. Prolog Systeme sind zur Zeit auf Mikrocomputern noch zu wenig ausgereift; Smalltalk ist auf Mikrocomputern praktisch (noch?) nicht anzutreffen.

Der einzige Grund, warum Logo überflüssig sein könnte, ist, daß es die Sprache Lisp gibt. Lisp ist die tragende Sprache der "Künstlichen Intelligenz". (Ich habe erhebliche Zweifel, ob der Begriff "Künstliche Intelligenz" gut gewählt ist; er scheint sich aber eingebürgert zu haben und nicht mehr ausrottbar zu sein). Logo ist ein Dialekt von Lisp. Im Vergleich zu Lisp ist der Einstieg in Logo jedoch sehr viel leichter. Dies hat hauptsächlich folgende Gründe:

- Logo ist sehr viel besser auf Mikrocomputer verfügbar als Lisp.

- Logo ist in den mir bekannten Versionen in sehr viel freundlichere Programmierumgebungen (Editor, Diskettenbetriebssysteme) eingebettet als Lisp.
- Logo hat eine sehr viel stärker an der Umgangssprache orientierte Syntax als Lisp. (Die Syntax von Lisp ist in weiten Bereichen spiegelbildlich zu der von FORTH).
- Die "Turtle"-Geometrie von Logo, die im allgemeinen kein Bestandteil von Lisp-Systemen ist, ermöglicht frühe Erfolgserlebnisse beim Programmieren.
- Alle wesentlichen Merkmale von Lisp sind in Logo (wenn auch gelegentlich in etwas abgemagerter und abgewandelter Form) vorhanden: Listenverarbeitung, Modularität und damit auch Strukturierbarkeit (insbesondere ein uneingeschränktes Funktionskonzept), Symbolverarbeitung (ein äußerst flexibles Variablenkonzept), Interaktivität.

Wie Lisp ist Logo eine sehr mächtige Sprache. Man kann die oben erwähnten "klassischen" Programmiersprachen sehr leicht und in sehr natürlicher Weise in Logo simulieren. Das heißt, man kann (auch wenn man es nicht unbedingt immer sollte) in Logo Programme schreiben, welche zum Beispiel exakt den Aufbau von BASIC- oder Pascal-Programmen widerspiegeln. Deshalb sind Lisp und Logo wie kaum eine andere Sprache zur Vermittlung, zum Studium und zur Erprobung fundamentaler sprachübergreifender Grundkonzepte des Programmierens und der Informatik geeignet.

Da Lisp die tragende Sprache der künstlichen Intelligenz ist, ermöglicht Logo als "anfängerfreundlicher" Dialekt von Lisp einen "weichen" Einstieg in die Problemkreise und die Programmiermethodik der künstlichen Intelligenz. Wer dies vorhat, darf allerdings nicht allzu viel Zeit damit vertun, daß er die Schildkröte (bzw. den Igel) kreuz und quer über den Bildschirm marschieren läßt, sondern er sollte sich gezielt dem in der populären Logo-Literatur oft vernachlässigten Themenkreis der Listen- und der Symbolverarbeitung zuwenden.

An diese Stelle ist auch ein Wort der Anerkennung angebracht. Existenz, Form, Inhalt und Stil dieses Buches verdanken der Unterstützung durch Freunde und Kollegen viel. Ohne die Mitwirkung von Hal Abelson und Leigh Klotz (Cambridge, Massachusetts), wäre es kaum zur Entwicklung des deutschen MIT Logo in der vorliegenden Form gekommen. Herbert Löthe (Kernen im Remstal) hat sich persönlich sehr stark für die Entwicklung des deutschen Logo eingesetzt. Ihm, Ulrich Hoppe (Stuttgart, Bad Cannstatt) und Werner Quehl (Esslingen) verdanke ich viele Anregungen aus Gesprächen und Diskussionen. Meine Frau hat die Rohfassung des Manuskripts gelesen und damit die Lesbarkeit des Textes erheblich verbessert. Meine Söhne Oliver und Bernd haben mit großer Ausdauer einen Teil der Beispiele des Buches ausprobiert. Gernot Dorn (Darmstadt), Martin Frodl (Reutlingen), Ernst Haas (Bad Urach) und Jens Jungbeck (Pliezhausen) haben das Manuskript studiert und mich auf Fehler und Verbesserungsmöglichkeiten aufmerksam gemacht. Besonders der Logo-Anfänger wird stark von ihren Verbesserungsvorschlägen profitieren. Alle verbleibenden Fehler und Unzulänglichkeiten gehen auf das Konto des Autors.

Reutlingen, September 1985 Jochen Ziegenbalg

Inhalt

Einleitende Bemerkungen zur Zielsetzung, Stilistik und Themenauswahl dieses Buches

Hinter dem Namen Logo steckt nicht nur eine Programmiersprache sondern auch eine "Philosophie" des Programmierens und Problemlösens mit Hilfe von Computern; also dessen, was man in den U.S.A. global als "computing" bezeichnet und wofür mir in der deutschen Sprache keine angemessene Übersetzung bekannt ist.

Ziel dieses Buches ist nicht so sehr, in eine bestimmte Logo-Version auf einem bestimmten Computer einzuführen, sondern durch die Einführung in die Sprachelemente und Methodologie von Logo etwas von dieser Art des Programmierens zu vermitteln. Deshalb ist dies auch nicht ausschließlich ein Logo-Buch. Vieles, was im folgenden exemplarisch am Beispiel der Sprache Logo entwickelt wird, ist sinngemäß auf andere Programmiersprachen übertragbar. Allerdings bieten andere Sprachen meist nicht das Handwerkszeug und die Flexibilität, um all das zu realisieren, was in Logo möglich ist.

Im Bereiche des Problemlösens durch die Programmierung von Computern kann man grob gesprochen folgende drei Vorgehensweisen feststellen:

(1.) Das ad hoc "Herunterhacken" eines Programmes.

(2.) Die systematische Entwicklung der kompletten Problemlösung mit Hilfe von "Papier und Bleistift" und die darauffolgende Übertragung der Problemlösung auf den Computer. Programmieren bedeutet in diesem Fall nicht viel mehr als routinehaftes Codieren.

(3.) Die Methode des aufbauenden Problemlösens. Dabei wird die Lösung eines komplexen Gesamtproblems in kleinen, überschaubaren Schritten entwickelt. Jeder Teilschritt wird möglichst unabhängig von den anderen bearbeitet und sofort nach seiner Bearbeitung ausgetestet.

Keine dieser Methoden wird von einer bestimmten Programmiersprache erzwungen; aber jede dieser Methoden wird von bestimmten Programmiersprachen unterstützt und provoziert.

Obwohl die in (1.) beschriebene Methode in praktisch allen "offiziellen" Stellungnahmen verschmäht wird, ist es die in der Praxis wohl am häufigsten anzutreffende Art des Arbeitens mit Computern. Dies hängt auch mit der Beschränktheit der heute anzutreffenden Heim-und Personal Computer zusammen.

Methode (2.) trennt die Phase der Problemlösung von der Phase ihrer Umsetzung auf einen Computer. Dies kommt der Mentalität mancher Methodiker entgegen, welche die "reine" Problemlösung so weit wie möglich von den niedrigen Aspekten der maschinellen Ausführbarkeit trennen möchten. Manch einer geht dabei so weit, zu betonen, daß im Unterrichtsrahmen die Umsetzung einer theoretisch erarbeiteten Problemlösung auf den Computer gar nicht mehr notwendig sei. Denn das Problem sei ja (theoretisch) gelöst, und das reiche ja aus. Programmieren fällt in dieser Sichtweise praktisch zusammen mit der niedrigen Tätigkeit des Codierens, also einer Tätigkeit, die nicht gerade das Bild von Kreativität hervorruft. Die Umsetzung einer mit Papier und Bleistift erarbeiteten Lösung in ein Computerprogramm

hat dabei gewisse Ähnlichkeiten mit dem Abtippen eines handgeschriebenen Manuskripts auf einer Schreibmaschine.

Die dritte Methode geht davon aus, daß es der menschlichen Natur am eheisten entspricht, wenn man es in einem Pröblemlöseprozeß jeweils mit kleinen, überschaubaren und gut kontrollierbaren Häppchen zu tun hat. Komplexere Probleme, die es natürlich auch gibt, sind dabei in eine ganze Hierarchie von kleineren, begrenzten Teilproblemen zu zerlegen. Die Lösungen der einzelnen Teilprobleme müssen sich gut zu einer Lösung des Gesamtproblems zusammensetzen lassen. Das heißt, daß die "Schnittstellen" der Teilprobleme sauber definiert sein müssen. Diese Vorgehensweise bezeichnet man gelegentlich als das "Baukastenprinzip"; in der Fachsprache der Informatik auch als Modularität.

Während Modularität grundsätzlich sowohl im Rahmen von Methode (2.) als auch von Methode (3.) eine Rolle spielt, unterscheiden sich beide Verfahrensweisen stark in der praktischen Realisierung. In Methode (2.) ist Modularität ein theoretisches Konzept; oft verbunden mit der Forderung nach der Anwendung des "Top-Down-Prinzips". Letzteres besagt, daß das Verfahren zur Lösung eines Problems von "oben nach unten" abzulaufen habe. Zuerst identifiziere man die gröbsten Teilprobleme des gegebenen Problems, dann deren größte Teilprobleme, und so weiter, bis man in einem sich baumartig verzweigenden Prozeß schließlich zu kleinen, überschaubaren Elementarproblemen kommt. Diese werden bearbeitet, die Einzellösungen werden zu Lösungen höherrangiger Teilprobleme und diese schließlich zur Lösung des Gesamtproblems zusammengesetzt.

Die strikte Forderung nach Anwendung des "Top-Down-Prinzips" hat jedoch etwas Dogmatisches an sich. Viele Problemlöseprozesse laufen auch von unten nach oben oder in Zickzackbewegungen ab. Ein Kind, das zum Beispiel mit Bauklötzen spielt, dürfte wohl kaum den kompletten Bauplan einer Burg vor Augen haben, wenn es Stein auf Stein setzend, verschiedene Realisierungsformen, wie zum Beispiel verschiedene Türmchen, verschiedene Tore und dergleichen ausprobiert und das, was ihm am besten gefällt, dann zur Burganlage zusammenbaut. Auch ist die Burg wohl nie in dem Sinne fertig, wie etwa ein Problem nach dem Top-Down-Prinzip mit seiner Lösung fertig und abgehakt ist. Das Kind wird wohl, solange es überhaupt mit der Burg spielt, immer wieder damit herumexperimentieren: einige neue Türme hinzufügen, einige ältere Gebäude entfernen oder abändern.

Dieses Bild des Arbeitens und Problemlösens liegt besonders der dritten oben beschriebenen Methode zugrunde. Man sollte diesen Arbeitsstil nicht geringschätzen. Nicht nur das spielerische Bauen von Burgen sondern auch "ernsthafte" Arbeit in Wissenschaft, Forschung und Entwicklung läuft häufig nach diesem Muster ab; wahrscheinlich sogar sehr viel häufiger als nach der etwas steril anmutenden Methode (2.).

Die Programmiersprache Logo ist so konstruiert, daß sie diesen offenen Stil des Problemlösens unterstützt. Spracheigenschaften, die diesem Ziele dienen, sind neben der oben erwähnten Modularität ein sehr hoher Grad an Interaktivität und die Erweiterbarkeit der Sprache. Für den Lernenden bedeutet Interaktivität, daß er schnell Rückmeldungen (in gut verstehbarer Form) bekommt und dadurch Fehler schnell erkennen kann; daß die Beseitigung von Fehlern sehr leicht gemacht wird

und daß ihm bestimmte effiziente Testhilfen zur Verfügung stehen. Eine wichtige Testhilfe ist zum Beispiel der "Protokoll"-Modus, in dem der Benutzer den Ablauf seines Programmes in Einzelschritten verfolgen kann.

Die Erweiterbarkeit einer Programmiersprache bedeutet schließlich, daß der Benutzer sich aus den Grundbefehlen der Sprache neue "Superbefehle" zusammenbauen und daß er darüberhinaus auch aus Grund- und Superbefehlen neue "Super-Superbefehle" definieren kann. Logo ist nicht die einzige Programmiersprache, in der das möglich ist, sondern Logo ist ein Dialekt der sehr viel älteren Sprache Lisp, für die das Merkmal der Erweiterbarkeit ebenfalls grundlegend ist. Auch die Sprache Forth gehört zu den erweiterbaren Sprachen.

Die Interaktivität von Logo macht es wünschenswert und sogar erforderlich, daß die behandelten Beispiele von Anfang an auf einem Computer ausprobiert werden. Es ist nicht sehr sinnvoll, Logo im "Trockenkurs" erlernen zu wollen. Es hat auch nicht viel Zweck, Logo-Prozeduren erst mit Papier und Bleistift aufzuschreiben und dann in den Computer zu übertragen. Denn Logo-Prozeduren sollten stets so klein und überschaubar gehalten werden, daß sie nur aus wenigen Zeilen bestehen und auf jeden Fall eine Bildschirmseite nicht überschreiten. Der Computerbildschirm übernimmt beim interaktiven Arbeiten mit Logo vielfach die traditionelle Rolle des Konzeptpapiers. Das direkte Arbeiten am Computer hat den Vorteil, daß Logo-Prozeduren sofort ausgetestet werden können, sobald sie geschrieben sind. Diese Möglichkeit, die Korrektheit der Programmentwicklung jederzeit durch Überprüfung des jeweils neu hinzugenommenen Moduls absichern zu können, ist besonders wichtig bei der Entwicklung größerer Programmpakete.

Man sollte also beim Lesen und Durcharbeiten dieses Buches einen Computer neben sich stehen haben, um die beschriebenen Sprachelemente und Beispiele sofort ausprobieren zu können. Man sollte sich auch nicht scheuen, mit sinnvoll erscheinenden Alternativen zu experimentieren. Keine Angst, solange Sie sich nicht mit "Hammer und Meißel" am Computer zu schaffen machen, können Sie kaum etwas zerstören. Das Arbeiten mit Logo hat sehr viel mit der Methode von Versuch und Irrtum zu tun.

Da dieses Buch nicht in ein spezielles Logo-System einführen will, kann es das Handbuch Ihrer Logo-Version nicht ersetzen. Letzteres ist vor allem auch unentbehrlich, wenn Sie einen Überblick über die komplette Befehlsliste Ihrer Logo-Version haben wollen. Bei der Behandlung einzelner Grundbefehle und anderer Sprachpartikel wurde hier besonderes Gewicht auf solche Elemente gelegt, die in den meisten Logo-Versionen (oft in ähnlicher Form) vorkommen und die für Logo typisch sind. Auch wenn Sie an speziellen technischen Details interessiert sind, müssen Sie im Handbuch Ihrer Logo-Version nachschlagen.

Ein weiteres Problem dieses Buches ergibt sich daraus, daß nicht alle Logo-Versionen identisch sind. Zum Beispiel ist Logo auch eine der ganz wenigen (wenn nicht die einzige) Programmiersprachen, die es mit deutschen Grundbefehlen und Fehlermeldungen gibt. Diesem Buch liegt primär die deutschsprachige MIT-Version von Logo zugrunde. Sie stimmt auch mit der beim IWT-Verlag erhältlichen deutschen Logo-Version für den Apple II und den Commodore 64 Computer überein.

Strukturell ist sie identisch mit den englischsprachigen Versionen der Firmen Krell und Terrapin für den Apple II Computer und mit dem Commodore Logo.

Weitere Logo-Versionen stammen von den Firmen LCSI ("Apple Logo"), TLC (unter CP/M 80) und Digital Research. An einigen syntaktisch besonders wichtigen Stellen wird explizit auf Unterschiede in den verschiedenen Logo-Versionen hingewiesen; besonders auf die Unterschiede zwischen dem MIT Logo und dem LCSI Logo.

Man sollte sich von diesen sprachlichen Unterschieden nicht entmutigen lassen. Strukturell stimmen alle Logo-Versionen viel stärker miteinander überein, als es auf den ersten Blick vielleicht aussieht. Die meisten Unterschiede zwischen verschiedenen Logo-Versionen spielen sich zudem nur an der Oberfläche ab. Und diese Oberfläche kann zum einen leicht uminterpretiert werden. Zum Beispiel ist es wirklich nicht schwierig, bei einer englischsprachigen Logo-Version den Befehl FORWARD mit dem deutschen Befehl VORWAERTS in Verbindung zu bringen.

Zum anderen, und dies ist besonders wichtig, kann eine bestimmte Logo-Oberfläche aber auch leicht modifiziert werden. Wem der Befehl PENUP als nicht akzeptabel erscheint, kann ihn leicht durch einen eigenen Befehl, zum Beispiel durch den Befehl STIFTHOCH erstzen. Solche Ersetzungsmöglichkeiten gibt es nicht nur bei simplen Befehlen, sondern auch im tieferliegenden Bereiche der Kontrollstrukturen von Logo. Auch die Abänderung des Verhaltens des Logo-Interpreters ist mit geringem Aufwand möglich. Beispiele hierzu werden später gegeben.

Schließlich einige Bemerkungen zur Themenauswahl. Besonders fazsinierend an Logo ist für den Anfänger der leichte Einstieg in den Themenkomplex der "Igel"-Geometrie (englisch: turtle geometry). Viele Einführungen in Logo beschäftigen sich deshalb sehr ausführlich mit diesem Themenkomplex. In manchen Büchern wird außer der Igel-Geometrie fast nichts behandelt. Dies hat dazu geführt, daß Logo gelegentlich als Graphik-Sprache angesehen wird.

Nun ist es zwar richtig, daß Logo über ein reichhaltiges Repertoire an Graphik-Befehlen verfügt, es wäre aber falsch, Logo als "Nur-Graphik-Sprache" hinzustellen. Im Gegenteil: die eigentliche Stärke von Logo, das was Logo (und Lisp) von den meisten anderen Programmiersprachen am gravierendsten unterscheidet, ist nicht die Graphik, sondern sind "Listenverarbeitung" und "funktionales Programmieren". Was genau darunter zu verstehen ist, wird später erläutert. Jedenfalls soll die Graphik hier in dem ihr gebührenden Rahmen und nicht so übergewichtig wie in den meisten populären Büchern über Logo behandelt werden.

Abschließend noch einige Hinweise zu den einzelnen Kapiteln dieses Buches. *Kapitel 1* dient der informellen Einführung in das interaktive Arbeiten mit Logo. Dieses Kapitel soll auch zu der Einsicht führen, daß das Arbeiten mit Computern eine fast ebenso selbstverständliche Angelegenheit sein kann wie etwa das Arbeiten mit einem Taschenrechner, vor dem ja wohl kaum jemand Angst haben dürfte.

Kapitel 2 führt ein in die Programmierung von Prozeduren und Funktionen. Es ist charakteristisch für das Arbeiten mit Lisp- und Logo-Systemen, daß dieser für die gesamte Informatik fundamentale Themenkreis gleich von Anfang an behandelt

wird; sehr viel früher als dies bei konventionellen Programmiersprachen üblicherweise der Fall ist.

In *Kapitel 3* werden einige Systemdetails behandelt. Diese Kenntnisse stellen ein "notwendiges Übel" dar, das auch irgendwann (und zwar nicht allzu spät) besprochen werden muß, wenn man das System in effizienter Weise nutzen will. Diese Art von Systemkenntnissen kann aber nicht als fundamental bezeichnet werden, da sich diese Details sehr schnell von Computergeneration zu Computergeneration ändern. Es ist eine Art von "instrumentellem" Wissen. Die Geläufigkeit im Umgang mit dem System kommt auch nicht vom Lesen sondern ist nur durch Übung zu erreichen.

In *Kapitel 4* werden die fundamentalen Datenstrukturen von Logo behandelt. Besonders bei diesem Thema unterscheiden sich Lisp und Logo sehr stark (und sehr vorteilhaft) von praktisch allen anderen Programmiersprachen. Zu den wesentlichen Merkmalen von Logo gehört die Unterscheidung zwischen dem Namen und dem Wert von Variablen und der dynamische und äußerst flexible Datentyp der Liste. Wer schon andere Programmiersprachen kennt, sollte hier besonders gründlich lesen.

In *Kapitel 5* wird systematischer auf die Kontrollstrukturen von Logo eingegangen, die bis dahin nur sporadisch behandelt worden sind. Es wird insbesondere darauf hingewiesen, daß auch Prozedur- und Funktionsaufrufe als Kontrollstrukturen anzusehen sind. Sehr ausführlich wird auf die Rekursion, die für Lisp und Logo fundamentale Kontrollstruktur, eingegangen. Der TUE-Befehl zur expliziten Auswertung von Listen ermöglicht schließlich solche Dinge wie die Übergabe von Funktionen und Programmen als Eingabeparameter, die Einführung benutzerdefinierter Kontrollstrukturen und Modifikationen des Interpreter-Verhaltens.

Die Kapitel 1 bis 5 (je einschließlich) stellen eine Art Grundwissen für das Programmieren in Logo dar. Die hier behandelten Beispiele wurden ad hoc zur Beschreibung der jeweiligen Sprachelemente herangezogen. Um die "Dürre" von ausschließlich syntaktischer Behandlung der Sprachelemente zu vermeiden, wurde versucht, jeweils anhand von kleinen, lokalen Beispielen aufzuzeigen, wie die behandelten Methoden mit Gewinn angewandt werden können. In diesen einführenden Kapiteln kann aber keine vertiefte Behandlung dieser Beispiele erfolgen.

In *Kapitel 6* wenden wir uns intensiv den Fragen der Programmierstilistik zu. Lokale Variable sollen zur "Sicherheit des Programmierens" beitragen. Sie erlauben besonders im Zusammenhang mit dem funktionalen Programmieren eine saubere Gestaltung der Schnittstellen zwischen den einzelnen Moduln. Die Art der von Logo praktizierten Variablensuche und -belegung erlaubt ein "kontextgebundenes" Programmieren mit lokalen Variablen. Verschiedene Methoden zur Erzeugung von lokalen Variablen werden diskutiert.

Die "tail recursion" (Endrekursion) ist eine spezielle Form der Rekursion, bei der kein Rekursionsstack aufgebaut werden muß. Endrekursive Prozeduren können deshalb im Gegensatz zu "voll" rekursiven Prozeduren im Prinzip beliebig lange laufen. Eine Methode zur Überführung gewisser rekursiver Funktionen in endrekrusive Varianten wird aufgezeigt.

Der zweite Teil dieses Kapitels dient einer beträchtlichen Vertiefung des Funktionskonzepts von Logo. Beispiele von Prozeduren bzw. Funktionen, deren Argumente selbst Funktionen sind, werden im Zusammenhang mit den folgenden Problemkreisen behandelt: die Erstellung von Wertetafeln, die Berechnung des Differenzenquotienten, die Funktionsverkettung (Produkte von Funktionen), das Potenzieren von Funktionen und das Arbeiten mit Funktionen, die als Rechenausdrücke gegeben sind.

Programmieren als Spracherweiterung ist eigentlich das Grundthema des gesamten Buches. In *Kapitel 7* wird aufgezeigt, in welch umfassender Weise Logo als erweiterbare Sprache konzipiert ist. Es ist nicht nur möglich, neue Prozeduren oder Funktionen zu programmieren, sondern die Funktionalität und das Listenkonzept von Logo ermöglichen es auch, neue Kontrollstrukturen in die Sprache einzugliedern. In den ersten Abschnitten dieses Kapitels wird gezeigt, wie man die folgenden typischen Kontrollstrukturen Algol-ähnlicher Sprachen in Logo nachbauen kann: SOLANGE (WHILE), WDH (REPEAT), FALL (CASE), FUER (FOR). Ein besonderer Vorteil wird dabei deutlich: da man diese Kontrollstrukturen selbst definiert hat, bewahrt man sich auch die volle Kontrolle darüber und kann sie eigenen Bedürfnissen optimal anpassen; zum Beispiel: FUER (FOR) mit oder ohne Schrittweite, FALL (CASE) mit oder ohne ANSONSTEN (OTHERWISE).

Wie man neue Datenstrukturen in Logo einführen kann, wird am Beispiel des Datentyps FELD (ARRAY) gezeigt. Erste Grundoperationen für den Umgang mit Feldern werden bereitgestellt. Als kleines Anwendungsbeispiel wird die Simulation eines Zufallsprozesses (das sogenannte Sammlerproblem) behandelt.

In *Kapitel 8* wird die interne Struktur von Prozeduren untersucht, die mit Hilfe der Logo-Funktion PRLISTE offengelegt wird. Die Tatsache, daß Prozeduren auch als Listen gespeichert werden und daß alle Listenoperationen auf diese Prozedurlisten anwendbar sind, ermöglicht die Veränderung von Prozeduren durch Prozeduren und die automatische Übersetzung (einschließlich der Neudefinition) von Prozeduren in einer verblüffenden Natürlichkeit und Eleganz. Als weitere nützliche Anwendung der PRLISTE-Funktion erhalten wir die Möglichkeit, prädikative Prozeduren GRUNDWORT? beziehungsweise PROZEDURNAME? zu schreiben, mit denen wir feststellen können, ob ein bestimmtes eingegebenes Wort ein Logo-Grundwort beziehungsweise ein Prozedurname ist oder nicht. Dadurch ergibt sich die Möglichkeit, vor dem Aufruf von Prozeduren festzustellen, ob sie sich überhaupt in der Arbeitsumgebung befinden. Ist dies nicht der Fall, so kann man sie von Diskette nachladen und dadurch eine etwaige Fehlermeldung mit anschließendem "Ausstieg" aus der Verarbeitungskette verhindern.

In *Kapitel 9* sind einige Dienstprogramme zusammengestellt, die dem Benutzer das "Programmierleben" gelegentlich sehr erleichtern können: der Quicksort, Engels Universalalgorithmus für eine Reihe mathematischer Standardfunktionen, Prozeduren für das stellengerechte Ausdrucken von Zahlen und für das "pretty printing" von Prozeduren.

Einige allgemeine Hinweise für das Arbeiten mit Computern und Logo

(1.) Irren ist menschlich. Dazu gehört, daß man sich gelegentlich vertippt. Es gibt verschiedene Methoden, um **Tippfehler** in Logo zu korrigieren. Sie werden in Kapitel 3, besonders Abschnitt 3.3 besprochen.

Hier vorab eine der einfachsten Möglichkeiten. Fast jeder Computer verfügt über eine "Links-Lösch-Taste". Wenn man sie drückt, wird das Zeichen links vom Cursor (Blinker) gelöscht. Diese Taste ist auf den meisten Tastaturen mit DEL (delete: löschen) bezeichnet. Auf dem Apple-Computer wird bei vielen Logo-Versionen (so zum Beispiel bei der MIT-Version und der Terrapin-Version) die ESC-Taste (Escape-Taste) als DEL-Taste verwendet.

(2.) Die Art und Weise, wie **Dezimalzahlen** vom Computer ausgedruckt werden, ist sehr stark vom benutzten System abhängig. Hier kann es auch Unterschiede zwischen verschiedenen Logo-Versionen geben, die auf demselben Computer laufen (auf dem Apple-Computer laufen zum Beispiel mindestens drei verschiedene Logo-Versionen).

Das Ergebnis der Division 6 / 2 wird dabei manchmal als 3. und manchmal als 3.0 dargestellt. Ebenso wird 2 / 4 manchmal als 0.5 aber auch nur als .5 dargestellt.

In diesem Buch werden die befriedigenderen Formen 3.0 und 0.5 verwendet.

Außerdem drucken alle Computer die Ergebnisse linksbündig, also nicht stellengerecht aus, falls keine besonderen Anweisungen gegeben werden. Linksbündig geschriebene Tabellen sind sehr schlecht lesbar. Im folgenden werden der besseren Lesbarkeit halber Ergebnisse in Tabellenform meist stellengerecht dargestellt.

In Kapitel 9, Abschnitt 9.3, findet der interessierte Leser Dienstprogramme, mit denen er einen stellengerechten Ausdruck erreichen kann.

(3.) Eine Bemerkung zum Begriff des **Prozentsatzes** bzw. Zinssatzes. Bei der Arbeit mit Computerprogrammen befindet man sich gelegentlich in der Situation, daß am Bildschirm die folgende Meldung erscheint:

GEBEN SIE DEN ZINSSATZ EIN:

Angenommen, der entsprechende Zinssatz ist 4.5 %. Aus eigener Erfahrung mit ähnlichen Programmen wissen Sie (oder werden es sehr bald wissen), daß der Computer im allgemeinen Eingaben wie "4.5 PROZENT" oder auch "4.5 %" nicht mag. Er erwartet in solchen Fällen praktisch immer eine Zahl ohne weitere Zusätze textlicher Art. Welche Zahl ist die Richtige? Man ist versucht, 4.5 einzugeben. Aber 4.5 % ist nicht dasselbe wie 4.5. Denn ist G eine feste Größe; z.B. eine Strecke, eine Zeitspanne, ein Kuchen, ein Guthaben, eine Menge Benzin, u.s.w., dann sind die folgenden sechs Bezeichnungen gleichwertig:

4.5 Prozent von G;
4.5 % von G;
4.5 Hundertstel von G;
G * 4.5 / 100;
G * 4.5 * 1 / 100   und
G * 0.045

Wenn man eine einzige Zahl einzugeben hat, dann ist die korrekte Zahl also 0.045. Nun wissen aber die Programmierer, daß viele Menschen beim Zinssatz von 4.5 % lieber 4.5 als 0.045 eingeben und korrigieren die eigentlich inkorrekte Eingabe (4.5), indem sie sie im weiteren Verlauf des Programms durch 100 teilen. Leider weiß der Benutzer eines solchen Programmes meist nicht, ob der Programmierer schon vorsorglich durch 100 dividiert hat. Der Programmierer könnte dies z.B. dadurch deutlich machen, daß er die Computermeldung etwas ausführlicher programmiert:

GEBEN SIE DEN ZINSSATZ EIN (IN HUNDERTSTELN):

In Hundertsteln ausgedrückt sind nun 4.5 % tatsächlich 4.5, denn 4.5 Hundertstel sind ja gerade 4.5 %.

In diesem Buch wird bei Ein- und Ausgaben stets die zweite, an sich weniger korrekte, in der Praxis aber gebräuchlichere Form verwendet; selbst dann, wenn dies im Computerdialog nicht explizit vermerkt sein sollte. Das heißt, Prozent- und Zinssätze werden stets in Hundertsteln ein- und ausgegeben.

Kapitel 1: Einstieg in das interaktive Arbeiten mit Logo

1.1 Logo als gewöhnlicher Taschenrechner

Wir können in Logo wie mit einem Taschenrechner arbeiten. Stellen wir uns einmal vor, wie wir die Rechnung 2+3 mit einem Taschenrechner ausführen:

- Zunächst geben wir die "Rechnung" 2+3 ein.
- Es tut sich nichts. Denn da der Taschenrechner nicht "wissen" kann, ob wir 2+3 oder vielleicht 2+3.1 oder sogar 2+3.14 ausrechnen wollen, müssen wir ihm sagen, wann die Eingabe der zweiten Zahl beendet ist.
- Dies tun wir, indem wir das Gleichheitszeichen " = " eingeben.
- Jetzt führt der Taschenrechner die Rechnung aus und stellt das Ergebnis 5 auf seiner kleinen Anzeige-Zeile dar.

In Logo geht es fast genauso. Wir müssen nur an Stelle des Gleichheitszeichens die RETURN-Taste drücken. Die RETURN-Taste wird immer dann benutzt, wenn die Eingabe einer Zahl, eines Rechenausdruckes oder eines Wortes abgeschlossen werden soll. An Stelle des Begriffs "RETURN-Taste" findet man häufig auch die Bezeichnung "CARRIAGE-RETURN-Taste" oder die Abkürzung "CR-Taste".

Probieren wir es einmal in Logo aus:

2.71+3.14
ERGEBNIS: 5.85

Wir können auch längere Ausdrücke eingeben. (Die Eingabe der letzten Zahl muß mit RETURN abgeschlossen werden).

1+2+3+4+5+6+7+8
ERGEBNIS: 36

Logo beherrscht die Punkt-vor-Strich-Rechnung:

2+3*4
ERGEBNIS: 14

Wir können auch Klammern setzen:

(2+3)*4
ERGEBNIS: 20

(100+1)*(100-1)
ERGEBNIS: 9999

Für die Division gibt es mehrere Möglichkeiten:
Erste Möglichkeit:

9/2
ERGEBNIS: 4.5

Das war zu erwarten.

Zweite Möglichkeit:

DIV 9 2 (lies: Division von 9 durch 2)
ERGEBNIS: 4

Probieren wir DIV noch einmal mit anderen Zahlen aus:

DIV 17 3
ERGEBNIS: 5

Einige weitere Versuche mit DIV lassen darauf schließen, daß DIV offenbar die Ganzzahldivision bewirkt. Zur Ganzzahldivision gehört immer die Frage nach dem Rest. Auch hierfür gibt es eine Funktion in Logo. Sie heißt natürlich REST.

REST 9 2 (lies: Rest bei der Division von 9 durch 2)
ERGEBNIS: 1

REST 17 3
ERGEBNIS: 2

Die **Division mit Rest** läßt sich gut veranschaulichen. Man stelle sich A als einen Stab der Länge 17, B als einen Stab der Länge 3 vor. Dann gibt DIV 17 3 an, wie oft der Dreierstab in den Siebzehnerstab geht; nämlich 5 mal. REST 17 3 gibt an, welcher Rest übrigbleibt, wenn wir den Siebzehnerstab solange um die Länge des Dreierstabes verkürzen, bis es nicht mehr geht. Wenn wir 5-mal die Länge des Dreierstabes vom Siebzehnerstab abschneiden, so bleibt zum Schluß ein Zweierstab übrig.

Diese geometrische Deutung ist in Abbildung 1.1 nochmals graphisch dargestellt. Man entnimmt ihr sofort die Gleichung:

$$17 = 3 * 5 + 2 \tag{1.1}$$

Dabei ist die Zahl 5 das Ergebnis von DIV 17 3 und die Zahl 2 das Ergebnis von REST 17 3.

Wir können Gleichung (1.1) also auch folgendermaßen schreiben:

$$17 = 3 * (\text{DIV } 17 \ 3) + \text{REST } 17 \ 3 \tag{1.2}$$

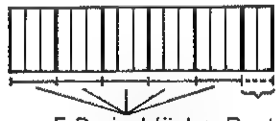


Abbildung 1.1: Division mit Rest

Eine kleine Anwendung: Die obigen Gleichungen (1.1) beziehungsweise (1.2) sind Beispiele für den **Satz von der Division mit Rest**. Er besagt, daß für je zwei beliebige natürliche Zahlen a und b ($b > 0$) stets die folgende Beziehung gilt:

$$a = b * (\text{DIV a b}) + \text{REST a b}$$

Wir können dies am Beispiel folgendermaßen überprüfen:

7 * (DIV 38 7) + REST 38 7
ERGEBNIS: 38

Die folgende in Logo gegebene Möglichkeit zur Probe ist besonders interessant. Wir werden uns im nächsten Abschnitt noch eingehender damit befassen.

38 = 7 * (DIV 38 7) + REST 38 7 (1.3)
ERGEBNIS: WAHR

47 = 9 * (DIV 47 8) + REST 47 8 (1.4)
ERGEBNIS: FALSCH

Wie jeder Taschenrechner verfügt Logo auch über einige numerische Standard-Funktionen. Diese variieren von System zu System. Einige davon seien in den nächsten Beispielen aufgezeigt:

Die Quadratwurzel-Funktion:

QW 6.25
ERGEBNIS: 2.5

Man beachte: Die Argumente (d.h. die Eingabewerte) von Funktionen werden einfach, durch Leerzeichen getrennt, hinter den Funktionsnamen geschrieben. Also: QW 6.25 und nicht etwa QW(6.25). Entsprechendes gilt auch für alle anderen Funktionen sowohl mit einem als auch mit mehreren Argumenten.

Die Sinus- und Cosinus-Funktion:

SIN 90
ERGEBNIS: 1

COS 60
ERGEBNIS: 0.5

Man beachte: Die Argumente von SIN und COS müssen im Gradmaß eingegeben werden.

Einige nützliche Funktionen zur Manipulation von Dezimalzahlen:

Die **Ganzteil-Funktion** INT gibt für positive Eingabewerte die größte ganze Zahl zurück, die kleiner oder gleich dem Eingabewert ist.

INT 6.28
ERGEBNIS: 6

Aber:

INT (-6.28)
ERGEBNIS: -6

Man beachte: Negative Eingabewerte sollten stets eingeklammert werden.

Die **Rundungs-Funktion** RUNDE gibt die dem Eingabewert am nächsten gelegene ganze Zahl zurück.

RUNDE 6.28
ERGEBNIS: 6

RUNDE 6.82
ERGEBNIS: 7

RUNDE (-6.28)
ERGEBNIS: -6

RUNDE (-6.82)
ERGEBNIS: -7

Probieren Sie RUNDE 3.5 und RUNDE (-3.5) aus.

Aufgabe 1.1: Stellen Sie die Funktionsgraphen von INT und RUNDE (von Hand) in einem x/y-Koordinatensystem dar.

1.2 Logo als "Taschenrechner" für Wahrheitswerte

Wie wir an den Gleichungen (1.3) und (1.4) gesehen haben, wertet Logo nicht nur numerische Funktionen sondern auch gewisse Aussagen aus.

Dazu einige weitere Beispiele:

2+3 = 5
ERGEBNIS: WAHR

2*3 = 7
ERGEBNIS: FALSCH

4 < 6
ERGEBNIS: WAHR

7 < 5
ERGEBNIS: FALSCH

Zur **Verneinung** von Aussagen dient das Logo-Grundwort NICHT (Apple II Logo) bzw. NICHT? (Commodore Logo):

NICHT 4 > 5 (bzw. Commodore-Version: NICHT? 4 > 5)
ERGEBNIS: WAHR

Man hätte (der Deutlichkeit halber) auch Klammern benutzen können: NICHT (4 < 5). Dies erweist sich besonders bei verschachtelten Aussagen als nützlich.

Wir können auch mehrere Aussagen miteinander verknüpfen: Das Grundwort ALLE? gibt WAHR zurück, wenn alle der eingegebenen Aussagen wahr sind, sonst gibt ALLE? FALSCH zurück.

ALLE? (5 < 2*3) (2*4 = 8)
ERGEBNIS: WAHR

ALLE? (3+4 = 7) (SIN 45 = 2.6)
ERGEBNIS: FALSCH

Das Logo-Grundwort EINES? gibt WAHR zurück, wenn (mindestens) eine der eingegebenen Aussagen wahr ist, sonst gibt EINES? FALSCH zurück.

EINES? (2 = 3) (3 > 1)
ERGEBNIS: WAHR

EINES? (8 < 6) (QW 4 = 5)
ERGEBNIS: FALSCH

1.3 Logo als "Graphik"-Taschenrechner

1.3.1 Die Igel-Graphik

Geben Sie den Befehl BILD ein. (Beachten Sie dabei, daß jede Eingabe mit der RETURN-Taste abgeschlossen werden muß). In der Mitte des Bildschirmes erscheint ein kleines Dreieck.

BILD

Der Igel geht in die Startposition:
Standort = Bildmitte
Richtung = "Norden"

Abbildung 1.2

Das Dreieck soll eine **Schildkröte** (englisch: **turtle**) oder, wie es in der deutschen Logo-Version heißt, einen **Igel** darstellen. Diesen Igel kann man nun auf dem Bildschirm herumkommandieren. Hierzu einige Beispiele:

VORWAERTS 80



Der Igel geht 80 Einzelschritte vorwärts (d.h. in seine "Blickrichtung")

Abbildung 1.3

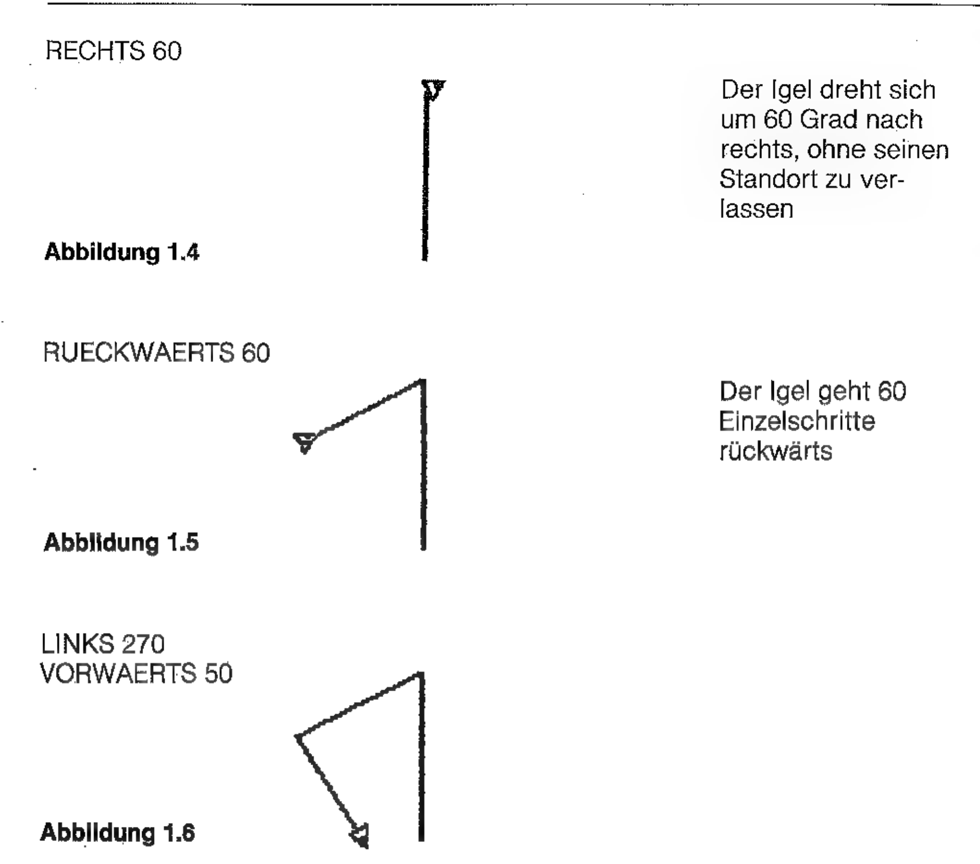


Abbildung 1.4

Abbildung 1.5

Abbildung 1.6

Experimentieren Sie weiter mit diesen Grundbefehlen. Man kann die Grund-Wörter auch abkürzen:

VW an Stelle von VORWAERTS
RW an Stelle von RUECKWAERTS
RE an Stelle von RECHTS
LI an Stelle von LINKS

Der Igel hat bisher bei jeder Bewegung seine Spur hinterlassen. Manchmal möchte man aber, daß er sich ohne Spur an eine andere Stelle des Bildschirms bewegt. Die Befehle

STIFTHOCH (kurz: SH) und
STIFTAB (kurz: SA)

ermöglichen das Umschalten zwischen den Bewegungsarten mit und ohne Spur.

Geben Sie jetzt wieder BILD ein, um den Bildschirm zu löschen und den Igel in die Ausgangsstellung (Mitte) zu bringen.

Gelegentlich möchte man eine Reihe gleichartiger Befehle hintereinander ausführen. Zum Beispiel

```
VW 80
RE  90
VW 80
RE  90
VW 80
RE  90
VW 80
RE  90
```

Wir hätten diese Befehlsfolge auch auf eine Zeile schreiben können:

```
VW 80  RE 90  VW 80  RE 90  VW 80  RE 90  VW 80  RE 90
```

(Nach der letzten 90 RETURN nicht vergessen).

Man beachte: Im folgenden Beispiel werden eckige Klammern benötigt. Wer eine Tastatur mit wählbarem Zeichensatz hat, muß den amerikanischen Zeichensatz (US ASCII) wählen. Diesen Zeichensatz sollte man immer eingeschaltet lassen, wenn man mit Logo arbeitet.

Wir haben oben viermal hintereinander die Befehlsfolge VW 80 RE 90 ausführen lassen. Da derartige Wiederholungen beim Arbeiten mit Computern häufig vorkommen, gibt es in Logo eine spezielle Wiederholungs-Anweisung. Sie heißt ganz einfach WIEDERHOLE (kurz: WH). WIEDERHOLE erwartet zwei Eingabewerte: der erste ist eine (natürliche) Zahl und gibt an, wie oft etwas wiederholt werden soll; der zweite gibt an, was zu wiederholen ist. Das zu Wiederholende muß in eckige Klammern geschrieben werden. Also im obigen Fall:

WIEDERHOLE 4 [VW 80 RE 90] beziehungsweise:
WH 4 [VW 80 RE 90]

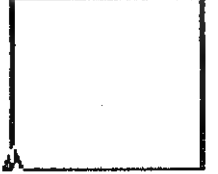


Abbildung 1.7: Quadrat

Als Bild erhalten wir ein Quadrat. Auf manchen Bildschirmen wird vielleicht auch ein Rechteck zu sehen sein. Das hängt vom Verhältnis zwischen dem horizontalen und dem vertikalen Abbildungsmaßstab des jeweiligen Bildschirmes ab. Man kann auf dreierlei Arten reagieren:

(1.) Man ignoriert das Problem; läßt das Rechteck Rechteck sein und stellt sich ein Quadrat darunter vor.

(2.) Wenn der Bildschirm über einen Knopf zur Regulierung der vertikalen Verzerrung verfügt, so dreht man an diesem Knopf so lange, bis aus dem Rechteck ein Quadrat geworden ist.

(3.) Man kann von Logo aus den vertikalen Verzerrungsmaßstab mit Hilfe des Befehles .SKALA verändern. Beispiele:

```
.SKALA 1.2
.SKALA 1
.SKALA 0.5
```

Der Voreinstellungswert beim MIT Logo für den Apple II Computer ist 0.8.

Vorsicht: Wenn Sie mit den später zu besprechenden "kartesischen" Graphik-Befehlen arbeiten, kann eine Änderung des .SKALA-Wertes unangenehme Auswirkungen haben.

Aufgabe 1.2: Zeichnen Sie jeweils ein möglichst großes regelmäßiges Dreieck, Fünfeck, Sechseck, Achteck, Sechsunddreißig-Eck, Hundert-Eck das noch ganz auf den Bildschirm paßt.

Man kann die Wiederholungs-Befehle auch ineinander verschachteln. Führen Sie die nächste Befehlsfolge zuerst "von Hand" mit Papier und Bleistift (Längenmaße in Millimetern) und dann mit dem Computer aus.

```
WH 36 [ WH 4 [ VW 50 RE 90 ] RE 10 ]
```

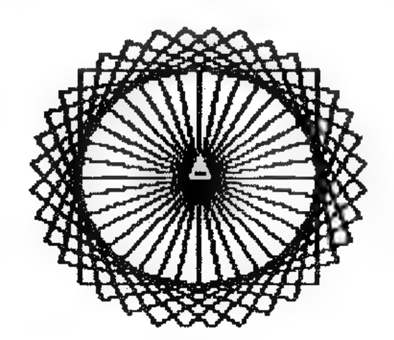


Abbildung 1.8: Rosette

Sieht das nicht hübsch aus? Zeichnen Sie weitere derartige Muster.

1.3.2 Einige Besonderheiten der Arbeit am Bildschirm

Geben Sie BILD, danach RECHTS 20 und schließlich VW 300 ein. Versuchen Sie, die Bewegungen des Igels am Bildschirm zu verfolgen.

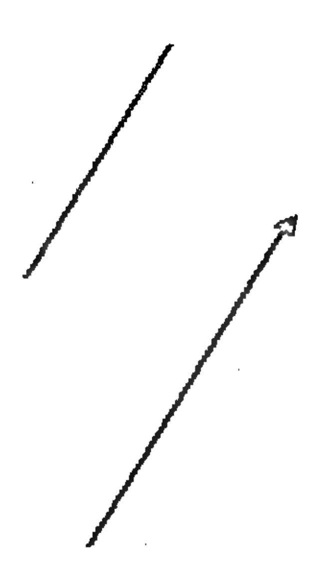


Abbildung 1.9

Der Igel bewegt sich von der Bildschirmmitte aus nach rechts oben. Da der Bildschirm für die auszuführende Bewegung nicht ausreicht, verläßt er ihn am oberen Rand, taucht am unteren Rand wieder auf und führt die vorgegebene Bewegung in der angegebenen Länge aus.

Tippen Sie jetzt die Taste Ctrl, halten Sie diese Taste gedrückt und tippen Sie nun gleichzeitig die Taste F. Im folgenden wird dieses Verfahren kurz mit "Ctrl-F tippen" bezeichnet. Die Ctrl-Taste (Ctrl steht für Control) ist eine Funktionstaste, ähnlich wie die Shift-Taste, also die "Hochstellungstaste". Man muß sie immer im Zusammenhang mit einer weiteren Taste drücken. Es kommt im allgemeinen nicht darauf an, ob der mit Ctrl eingegebene Buchstabe als Groß- oder als Kleinbuchstabe eingegeben wird; das heißt Ctrl-F und Ctrl-f bewirken dasselbe.

Nach Ctrl-F weitet sich der Bildschirm nach unten aus. Wir sehen, daß der Igel am unteren Bildschirmrand in horizontaler Richtung an derselben Stelle eintritt, an der er den Schirm am oberen Rand verlassen hat. Man kann sich das auch so vorstellen, als ob der obere Rand des Bildschirms ımit dem unteren Rand wie ein Zylinder zusammengeklebt ist. Was für oben und unten gilt, gilt entprechend auch für rechts und links. Der Bildschirm entspricht also einem Zylinder, dessen beide kreisförmigen Randlinien miteinander verklebt sind; also einem Autoreifen ohne Ventil (auch Torus genannt). Versuchen Sie die Befehlsfolge

RE 60 VW 400

Den soeben beschriebenen Bildschirm-Modus nennt man den Randsprung-Modus. Man kann den Bildschirm aber auch so einstellen, daß das Verlassen des

Bildschirmes unmöglich gemacht wird. Dieser zweite Modus heißt der Rand-Modus. Das Hin- und Herschalten zwische Randsprung-Modus und Rand-Modus geschieht mit den Logo-Befehlen RANDSPRUNG beziehungsweise RAND. Will der Igel im Rand-Modus den Bildschirmrand überschreiten, so erscheint die Fehlermeldung "IGEL IM AUS".

In englischsprachigen Logo-Versionen heißen die Befehle

RANDSPRUNG	WRAP	und
RAND	NOWRAP	

In manchen englischsprachigen Logo-Versionen darf der Igel im NOWRAP-Modus den Bildschirm verlassen. Man sieht ihn dann nur eine Weile lang nicht, bis er im Laufe seiner Reise wieder in den Bildschirmbereich eintritt. Derartige Systeme verfügen dann aber häufig noch über den Modus FENCE, der ähnlich wie der oben beschriebene RAND-Modus arbeitet.

Als wir mit Hilfe des Befehles BILD erstmalig vom Text-Bildschirm in den Graphik-Bildschirm umschalteten, standen uns am unteren Bildschirmrand noch vier Textzeilen zur Verfügung. Dort konnten wir überprüfen, ob wir alles richtig eingetippt hatten, und wir konnten dort auch etwaige Reaktionen, zum Beispiel Fehlermeldungen, des Computers zur Kenntnis nehmen. Kurz: die unteren vier Textzeilen dienten dem "Dialog mit dem Computer". Dieser Bildschirm-Modus wird als der Teilbild-Modus (englisch: splitscreen mode) bezeichnet; im Gegensatz zum Vollbild-Modus, in den wir mit Hilfe von Ctrl-F umgeschaltet haben. Insgesamt gibt es die folgenden Bildschirm-Betriebsarten:

deutsch:	englisch:	Befehle zum Umschalten:	
Teilbild	splitscreen	TEILBILD	bzw. Ctrl-S bzw. F3
Vollbild	fullscreen	VOLLBILD	bzw. Ctrl-F bzw. F5
Text	text		Ctrl-T bzw. F1

Man kann sich die Control-Komandos gut anhand der Anfangsbuchstaben der englischen Bezeichnungen merken.

Beim Commodore 64 Computer sind an Stelle von Ctrl-S, Ctrl-F und Ctrl-T die Funktionstasten F3, F5 und F1 zu benutzen.

Auch der Befehl LOESCHESCHIRM (kurz: LS) bewirkt neben dem Löschen des Bildschirms ein Umschalten in den Text-Modus.

Wenn beim Arbeiten im Vollbild-Modus ein (syntaktischer) Fehler auftritt, schaltet Logo automatisch vor dem Ausdrucken der Fehlermeldung in den Teilbild-Modus um.

1.3.3 Die kartesische Koordinaten-Graphik

Wenn man mit den oben beschriebenen Befehlen wie VORWAERTS, RUECKWAERTS, RECHTS oder LINKS, arbeitet, dann sollte man sich am besten in die Rolle des Igels versetzen und seinen Weg im Geiste vorausdenken oder nachvollziehen.

Man könnte diese Art der Geometrie als **beobachter-zentrierte Geometrie** bezeichnen. Es ist eine sehr naheliegende Art, sich geometrisch auszudrücken. Man denke etwa daran, wie üblicherweise meist der Weg zu bestimmten Zielen beschrieben wird: "Fahren Sie etwa 1 km geradeaus, dann geht es nach rechts, dann nach ungefähr 100 m nach links und dann noch knapp 500 m". Eine solche "Geometrie" versteht auch jemand, der kein formales geometrisches Wissen besitzt.

Für manche geometrischen Fragestellungen ist aber die **kartesische Koordinatengeometrie** (mit x- und y-Achse) besser geeignet. Dies ist besonders dann der Fall, wenn geometrische Darstellungsweisen dazu benutzt werden, um algebraische Sachverhalte zu veranschaulichen. In der Schulmathematik ist dies besonders bei den Themenbereichen "Kurven, Funktionsschaubilder, Histogramme (Stäbchendiagramme)" der Fall.

Logo verfügt auch über Grundbefehle der kartesischen Geometrie. Im Graphik-Modus (Vollbild) des Apple-Computers ist der Bildschirm zum Beispiel folgendermaßen begrenzt:

der linke Rand hat die x-Koordinate	-140
der rechte Rand hat die x-Koordinate	139
der untere Rand hat die y-Koordinate	-119
der obere Rand hat die y-Koordinate	120

Der Ursprung des Koordinatensystems liegt also ziemlich genau in der Mitte des Bildschirms.

Lassen Sie uns mit einigen Grundbefehlen experimentieren. Wenn man mit den kartesischen Befehlen arbeitet, ist es oft nicht sinnvoll, daß der Igel auf dem Bildschirm dargestellt wird. Man bringt ihn durch den Befehl VERSTECKIGEL (kurz: VI) zum Verschwinden und durch den Befehl ZEIGIGEL (kurz: ZI) wieder zum Vorschein. Aber auch wenn man den Igel nicht sieht, ist er dennoch da.

Führen Sie die nächsten Übungen mit sichtbarem und danach mit unsichtbarem Igel aus.

BILD
Der Igel steht in der Mitte des Bildschirms mit "Blickrichtung" nach oben (Norden).

AUFXY 40 60
Der Igel wird auf den Punkt mit der x-Koordinate 40 und der y-Koordinate 60 gesetzt. Da der "Stift" abgesetzt ist, sehen wir die Spur.

AUFX (-100)
Der Igel geht waagerecht ohne den y-Wert zu verändern auf den Punkt (-100 / 60).

AUFY 105
Der Igel wandert senkrecht nach oben auf den Punkt (-100 / 105). Er sollte jetzt den folgenden Weg zurückgelegt haben:

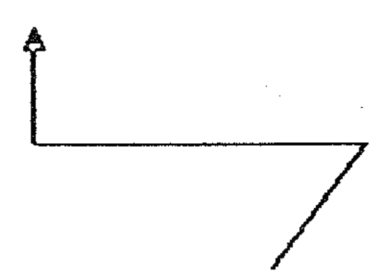


Abbildung 1.10

Wenn wir im Verlaufe einer komplizierten Bewegungsabfolge vergessen haben, auf welchen Standort sich der Igel genau befindet, so liefert uns der Befehl XKO den Wert seiner x- und YKO den Wert seiner y-Koordinate.

Welche Auswirkung wird die Befehlsfolge AUFXY (XKO + 40) (YKO + 20) haben? Probieren Sie es aus. Wie Sie sehen, sind damit relative Bewegungen gut beschreibbar.

Aufgabe 1.3: Zeichnen Sie einen reckteckigen "Kasten" mit den Eckpunkten auf den Koordinaten (20 / 30), (20 / 80), (110 / 80) und (110 / 30).

Aufgabe 1.4: Geben Sie eine Befehlsfolge zur Spiegelung des Igelpunktes an der ersten Winkelhalbierenden an.

1.3.4 Brücken zwischen Igel-Graphik und Koordinaten-Graphik

Geben Sie zunächst BILD ein. Der Igel blickt nach oben (Norden). Wir experimentieren weiter.

AUFKURS 90
Der Igel blickt nach rechts (Osten). Er verfügt über eine innere Windrose, die zwischen den Werten 0 und 360 Grad variiert.

0 bedeutet Norden
90 bedeutet Osten
180 bedeutet Süden
270 bedeutet Westen
360 bedeutet wieder Norden (0 = 360 als Windrosenrichtung)

AUFKURS (kurz: AK) hat zur Folge, daß sich der Igel in die angegebene Richtung dreht.

Geben Sie KURS ein (nach AUFKURS 90):
ERGEBNIS: 90.0

Der Befehl KURS gibt die Richtung (als Dezimalzahl) zurück, in die der Igel gerade blickt.

Geben Sie ein:

RICHTUNG 30 50
ERGEBNIS: 30.9632

Der Befehl RICHTUNG (kurz: RI) bewirkt, daß die Peilrichtung vom gegenwärtigen Standort des Igels zum Punkt mit den angegebenen Koordinaten (in diesem Fall (30 / 50)) als Windrosen-Wert zurückgegeben wird. In diese Richtung müßte sich der Igel drehen, wenn er den angegebenen Punkt (hier (30 / 50)) anpeilen wollte. Die Orientierung des Igels verändert sich durch den Befehl RICHTUNG jedoch noch nicht. Dies wird zum Beispiel durch die Befehlsfolge AUFKURS 30.9632 erreicht. Da RICHTUNG 30 50 gerade den Wert 30.9632 liefert, kann (und sollte) man in diesem Fall an Stelle von AUFKURS 30.9632 aber besser AUFKURS RICHTUNG 30 50 schreiben. Diese Befehlsfolge wird so abgearbeitet, daß zuerst der Wert von RICHTUNG 30 50 ermittelt und dann in den Befehl AUFKURS eingespeist wird. Wir können dies zum Beispiel folgendermaßen überprüfen:

```
BILD  AUFKURS  30.9632  VORWAERTS  80
MITTE  AUFKURS  RICHTUNG  30  50  VORWAERTS  80
```

Der Befehl MITTE bewirkt, daß der Igel in die Startstellung (Position: Bildschirmmitte; Richtung: Norden) gesetzt wird. Im Unterschied zu BILD wird bei MITTE jedoch der Bildschirm vorher nicht gelöscht.

Testen Sie jetzt:

```
AUFKURS  RICHTUNG  100  10
VORWAERTS  100
```

Probieren Sie die folgende hübsche Anwendung der Befehle AUFKURS und RICHTUNG aus:

```
WIEDERHOLE  1000 [ AUFKURS  RICHTUNG 0 0  RECHTS 90  VORWAERTS 2 ]
```

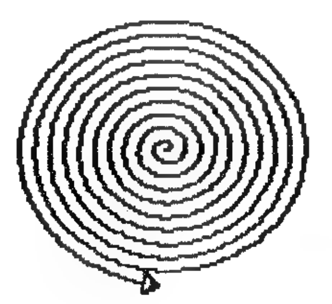


Abbildung 1.11: Spirale

Wir haben jetzt die wichtigsten Grundbefehle der Logo-Graphik kennengelernt. Einige weitere Befehle, wie FARBE und HINTERGRUND sind stark geräteabhängig. Ebenso der Befehl IGELZUSTAND : beim MIT Logo gibt er eine Liste bestehend aus vier Angaben zurück. Die Angaben haben die folgende Bedeutung:

1. Angabe: WAHR, wenn der Stift abgesetzt ist;
FALSCH, wenn der Stift hoch ist

2. Angabe: WAHR, wenn der Igel zu sehen ist;
FALSCH, wenn der Igel unsichtbar ist

3. Angabe: Hintergrundfarbe (siehe Handbuch)

4. Angabe: Farbe des Zeichenstifts (siehe Handbuch).

Abschließend seien noch zwei Hilfsbefehle erwähnt, die man gelegentlich braucht: LOESCHEBILD (kurz: LB) löscht den Graphik-Bildschirm, ohne die Stellung des Igels zu verändern. LOESCHESCHIRM (kurz: LS) löscht den Text-Bildschirm und setzt die *Schreibmarke* (englisch: *cursor*), auch *Blinker* genannt (selbst wenn er manchmal nicht blinkt), in die linke obere Ecke, die sogenannte *Home-Position des Text-Modus*. (Die *Home-Position des Graphik-Modus* ist der Koordinatenursprung (0/0)). Im Graphik-Modus bewirkt LOESCHESCHIRM darüberhinaus, daß Logo von diesem Modus in den Text-Modus zurückschaltet.

Kapitel 2: Prozeduren und Funktionen als "Superbefehle"

2.1 Einfachste Prozeduren im Graphik-Bereich

Im vorigen Kapitel haben wir gesehen, daß man mit der Befehlsfolge

VW 50 RE 90 VW 50 RE 90 VW 50 RE 90 VW 50 RE 90

beziehungsweise

WIEDERHOLE 4 [VORWAERTS 50 RECHTS 90] (2.1)

ein Quadrat zeichnen kann. Es ist nur natürlich, diese Befehlsfolge auch unter dem Namen QUADRAT zusammenzufassen und durch diesen Namen aufzurufen. Tippen Sie folgendes ein:

LERNE QUADRAT

Logo schaltet nun den Bildschirm um vom **Direktausführungs-Modus** ("Taschenrechner-Modus") in den Lern- oder **Editier-Modus**. Der Editier-Modus ist eine Art "Logo-Konzeptpapier". Man kann in ihm Texte aller Art erstellen: Notizen, Briefe und eben auch Programme. Der Hauptzweck des Logo-Editors ist jedoch die Erstellung von Programmen (beziehungsweise Prozeduren und Funktionen).

Tippen Sie jetzt die in (2.1) dargestellte Zeile und danach das Wort ENDE ein. Der Bildschirm sollte folgendermaßen aussehen:

```
PR QUADRAT
WIEDERHOLE 4 [ VORWAERTS 50  RECHTS 90 ]
ENDE
```

Die Buchstaben PR stehen für Prozedur. QUADRAT ist ein Beispiel für eine sehr einfache Logo-Prozedur

Bis jetzt ist noch nichts passiert. Logo weiß nicht, ob Sie mit dem Editieren aufhören wollen oder nicht. Das Wort ENDE bedeutet nur, daß die Prozedur QUADRAT abgeschlossen ist. Es könnte aber sein, daß jemand gleich anschließend an diese Prozedur eine weitere Prozedur, vielleicht die Prozedur DREIECK, aufschreiben möchte. Achten Sie auf die unterste Zeile am Bildschirm. Sie sagt Ihnen, wie Sie den Editier-Modus verlassen können (ohne den Rechner abzuschalten). Wenn Logo die aufgeschriebene(n) Prozedur(en) "lernen" soll, so muß Ctrl-C eingegeben werden. Beim Commodore 64 tut es auch die RUN/STOP-Taste. Manchmal hat man aber auch einfach Unsinn eingetippt, den man nicht konservieren lassen möchte. Will man den Editor verlassen, ohne daß Logo die editierte(n) Prozedur(en) lernen soll, so gibt man Ctrl-G ein.

Wir möchten allerdings unser QUADRAT lernen lassen. Also geben wir Ctrl-C ein. Logo reagiert nun so, daß es erstens wieder in den Direktausführungs-Modus zurückschaltet und dann die Meldung "QUADRAT GELERNT" bringt.

Probieren wir es aus! Tippen Sie QUADRAT ein. Tatsächlich, es funktioniert. Logo hat einen neuen Befehl gelernt. Den Namen haben wir zwar willkürlich, aber sinn-

voll vergeben. Wir hätten auch einfach Q oder HUGO oder X17E35 als Namen verwenden können; aber QUADRAT sagt dem Benutzer eben am deutlichsten, was der Befehl tut.

Mit der Befehlsfolge

WH 36 [WH 4 [VW40 RE90] RE10] (2.2)

aus Kapitel 1 konnten wir ein rosettenartiges Ornament zeichnen.

Aufgabe 2.1: Schreiben Sie entsprechend der Prozedur QUADRAT eine Prozedur namens ROSETTE, welche dieselbe Figur wie die Befehlsfolge (2.2) auf den Bildschirm zeichnet.

Nachdem wir die Prozedur QUADRAT definiert hatten, hätten wir im Direktausführungs-Modus an Stelle von (2.2) auch folgende Befehlsfolge verwenden können:

WH 36 [QUADRAT RE 10] (2.3)

Dies ist besser lesbar, nicht wahr? Schließlich können wir QUADRAT auch als Baustein in unserer ROSETTE verwenden:

```
PR ROSETTE
 WIEDERHOLE 36 [ QUADRAT RECHTS 10 ]
ENDE
```

Wir haben soeben ein erstes, winziges Beispiel für baukastenartiges, modulares Arbeiten kennengelernt.

Ein kleiner Hinweis zur Verwendung von Abkürzungen für Logo-Grundwörter in diesem Buch: In Beispielen, wo der interaktive Betrieb dargestellt wird, werden meist die Abkürzungen, in Prozeduren dagegen wird der besseren Lesbarkeit halber meist der volle Name verwendet. Außerdem werden wir der größeren Übersichtlichkeit halber im folgenden die zwischen PR und ENDE stehenden Zeilen jeweils wie im letzten Beispiel um eine Leerstelle einrücken.

Aufgabe 2.2: Schreiben Sie Prozeduren zum Zeichnen regelmäßiger Dreiecke, Fünfecke, Sechsecke, Hundertecke. Verwenden Sie diese Prozeduren im "interaktiven" Dialogbetrieb und andererseits als Bausteine in komplexeren Prozeduren.

2.2 Prozeduren mit Eingabeparametern

Die Definition der Prozedur QUADRAT stellt einen großen Fortschritt dar. Aber die Quadrate, die QUADRAT zeichnet sind alle gleichgroß, wenn auch möglicherweise in unterschiedlicher Lage. Eigentlich hätte die Prozedur nicht QUADRAT sondern QUADRAT50 heißen müssen, da sie nur Quadrate der Seitenlänge 50 zeichnen kann. Um ein Quadrat der Seitenlänge 40 zu zeichnen, könnten wir natürlich auch noch eine Prozedur QUADRAT40 definieren. Bald würden wir sicher noch hunderte weiterer Quadrat-Prozeduren benötigen, um Quadrate der verschiedensten Seitenlängen zeichnen zu können.

Es gibt eine bessere Lösung für unser Problem, Quadrate mit verschiedenen Seitenlängen zu zeichnen. Wir müssen der Prozedur QUADRAT einen Eingabeparameter beifügen, der beim Aufruf der Prozedur beliebig "bestückt" werden kann. Ein solcher Eingabeparameter ist so etwas wie eine Variable in der Prozedur QUADRAT. Aus bestimmten Gründen, die später im Kapitel über Namen näher zu erläutern sein werden, muß jeder solchen Variablen ein Doppelpunkt vorangestellt sein.

Definieren wir die folgende Prozedur:

```
PR QUADRAT :X
 WIEDERHOLE 4 [ VORWAERTS :X  RECHTS 90 ]
ENDE
```

Durch diese erneute Definition der Prozedur QUADRAT wird übrigens die vorherige Version überschrieben. Sie existiert nicht mehr.

Probieren Sie:

QUADRAT RETURN-Taste
Logo's Reaktion:
QUADRAT BRAUCHT MEHR EINGABEN (beim deutschen MIT Logo für den Apple II)
QUADRAT WILL MEHR DATEN (beim deutschen Commodore Logo)

Sehr sinnvoll! Da wir die Prozedur QUADRAT mit einem Eingabeparameter versehen haben, brauchen wir uns nicht zu wundern, daß Logo uns auf den fehlenden Eingabewert aufmerksam macht.

Versuchen wir es also noch einmal:

BILD
QUADRAT 50
QUADRAT 40
QUADRAT 30
QUADRAT 20
QUADRAT 10
QUADRAT 60
QUADRAT 70
QUADRAT 80
QUADRAT 90
QUADRAT 100

Nach jedem RETURN zeichnet der Igel ein Quadrat der angegebenen Größe. Insgesamt sieht die Figur jetzt folgendermaßen aus:

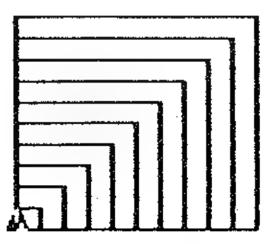


Abbildung 2.1: Quadrate

Bei der Ausführung von QUADRAT 40 ersetzt Logo den in der Definition verwendeten Parameter :X durch den aktuellen Wert 40. Man bezeichnet :X als einen **formalen** und 40 als einen **aktuellen** Parameter.
Ein formaler Parameter ist so etwas wie eine Variable oder eine Leerstelle, die zur Definitionszeit einer Prozedur offengehalten wird und die beim Lauf der Prozedur mit einem konkreten Wert ausgefüllt wird.

Die Bezeichnung des formalen Parameters war dabei ebenso willkürlich wie die Wahl des Prozedurnamens QUADRAT. An Stelle von :X hätten wir auch :SEITE oder :SEITENLAENGE oder :R25SY4 verwenden können.

Aufgabe 2.3: Versehen Sie auch die Prozedur ROSETTE mit einem formalen Parameter, der es Ihnen erlaubt, Rosetten in verschiedenen Größen zu zeichnen und experimentieren Sie mit dieser Prozedur.

Auch diese Version von ROSETTE befriedigt uns nach einiger Zeit des Experimentierens nicht mehr. Wir würden gern die Anzahl der "Blätter", also der Wiederholungen, variabel halten. Dazu müssen wir einen zweiten Eingabeparameter einführen. Die folgende Prozedur zeigt eine Möglichkeit. (Man kann jeder Prozedur beliebig viele Eingabeparameter hinzufügen, die man der Reihe nach, jeweils durch eine Leerstelle getrennt, hinter den Prozedurnamen schreibt).

```
PR ROSETTE :SEITE :N
 WIEDERHOLE :N [ QUADRAT :SEITE  RECHTS 360 / :N ]
ENDE
```

Nach jedem Zeichnen eines Quadrates dreht sich der Igel um 360/:N Grad nach rechts. Nach :N solchen Drehungen macht die Summe seiner Drehungen einen Vollkreis aus und die Rosette schließt sich.

Wir hätten als Seitenlängen-Parameter in ROSETTE auch :X an Stelle von :SEITE verwenden können, ohne daß das :X in QUADRAT und das :X in ROSETTE miteinander in Konflikt geraten:

```
PR ROSETTE :X :N
 WIEDERHOLE :N [ QUADRAT :X  RECHTS 360 / :N ]
ENDE
```

Nicht zum Ziel führt dagegen der folgende Versuch:

```
PR ROSETTE :SEITE :N
 WIEDERHOLE :N [ QUADRAT :X  RECHTS 360 / :N ]
ENDE
```

Denn ROSETTE sollte den Parameter :X beim Aufruf von QUADRAT mit einem konkreten Wert bestücken. ROSETTE kennt den Parameter :X aber gar nicht, da :X nicht unter den Eingabeparametern von ROSETTE vorkommt. Deshalb kann ROSETTE dem Parameter :X auch keinen Wert zuweisen. (Man sagt auch, der Parameter :X sei beim Aufruf von ROSETTE nicht belegt).

Aufgabe 2.4: Schreiben Sie eine Prozedur namens GLEICHSEITIGES.DREIECK :S zum Zeichnen eines gleichseitigen Dreiecks.

Hinweis: In Prozedurnamen darf auch der Punkt verwendet werden. Man erreicht dadurch eine bessere Lesbarkeit. Auch einige weitere Sonderzeichen, wie zum Beispiel das Zeichen # dürfen in Namen vorkommen. Spezielle Sonderzeichen, die man nicht in Prozedurnamen einbetten kann (beziehungsweise sollte) sind alle Arten von Klammern, und die Anführungszeichen. Jeder Name muß aber aus einer zusammenhängenden Zeichenkette bestehen; eine Einbettung von Leerzeichen (englisch: blanks) in Prozedur- oder sonstige Namen ist also nicht möglich.

Aufgabe 2.5: Schreiben Sie eine Prozedur POLYGON :N :S , mit der Sie regelmäßige Vielecke der Eckenzahl :N und Seitenlänge :S zeichnen können.

Probieren Sie aus POLYGON 100 2 aus.

Aufgabe 2.6: Schreiben Sie eine Prozedur KREIS, mit der Sie Kreise verschiedener Größe zeichnen können.

Aufgabe 2.7: Schreiben Sie eine Prozedur RECHTECK :A :B

Aufgabe 2.8: Schreiben Sie eine Prozedur PARALLELOGRAMM :A :B :W (mit den Seitenlängen :A und :B und dem Winkel :W als Innenwinkel beim Startpunkt).

Manchmal kann man einfachere Prozeduren als Spezialfälle von komplexeren Prozeduren darstellen. Zum Beispiel:

```
PR RECHTECK  :A  :B
 PARALLELOGRAMM  :A :B 90
ENDE
```

```
PR QUADRAT  :A
 RECHTECK  :A  :A
ENDE
```

Diese Elementarprozeduren können nun genutzt werden, um komplexere Bilder aufzubauen. Ein Beispiel:

```
PR KIRCHE  :G
 QUADRAT  :G
 VORWAERTS  :G
 RECHTS  30
 GLEICHSEITIGES.DREIECK  :G
 RECHTS  60
 VORWAERTS  :G
 PARALLELOGRAMM  :G*2  :G*0.4  120
 RECHTECK  :G*2  :G
 RUECKWAERTS  :G
 LINKS  90
 RUECKWAERTS  :G
ENDE
```

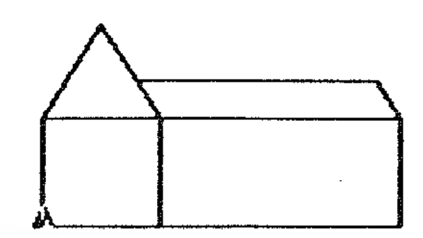


Abbildung 2.2: Kirche

Die letzten drei Befehle dienen dem Zweck, den Igel zum Schluß wieder auf den Anfangspunkt zurückzuführen. Dies erweist sich besonders dann als günstig, wenn man verschiedene Bilder zusammensetzen will.

Aufgabe 2.9: Schreiben Sie eine Gruppe von Prozeduren zum Thema "Dorf"; zum Beispiel HAUS, KIRCHE, STRASSE.MIT.HAEUSERN, MARKTPLATZ, ...
Versuchen Sie, entsprechend dem Baukastenprinzip auf bereits vorhandenen Prozeduren aufzubauen.

So kann man ein stilisiertes Blatt zeichnen:

```
PR BLATT
 WIEDERHOLE 60 [ VORWAERTS 2 RECHTS 1 ]
 RECHTS 120
 WIEDERHOLE 60 [ VORWAERTS 2 RECHTS 1 ]
 RECHTS 120
ENDE
```

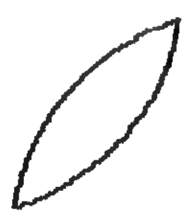


Abbildung 2.3: Blatt

Erweitern Sie die Prozedur BLATT so, daß man Blätter unterschiedlicher Größe zeichnen kann.

Aufgabe 2.10: Schreiben Sie eine Gruppe von Prozeduren zum Thema "Garten": BLUETE, BLUME, BLUMENWIESE, ... (unter Zuhilfenahme der obigen Prozedur BLATT).

Aufgabe 2.11: Schreiben Sie eine Gruppe von Prozeduren zum Thema "Wald": BLATT, ZWEIG, BAUM, WALD, ...

2.3 Prozeduren im nichtgraphischen Bereich

Prozeduren sind natürlich nicht nur im Kontext der Igel-Geometrie möglich. In der folgenden Prozedur wird zum Beispiel der Mittelwert der beiden Eingabewerte :A und :B ausgedruckt.

```
PR MITTELWERT  :A  :B
 DRUCKE  (:A + :B) / 2
ENDE
```

Dies ist aber kein Beispiel für guten Programmierstil, wie wir im Abschnitt über Funktionen gleich sehen werden.

Ein anderes Beispiel im Zusammenhang mit dem Druckbefehl:

Der Befehl DRUCKE [XZY 3.14 HALLO] führt dazu, daß Logo alles unverändert ausdruckt, was in den eckigen Klammern steht.

Wir können also zum Beispiel folgende Begrüßungsprozeduren schreiben:

```
PR FREUNDLICHE.BEGRUESSUNG
 DRUCKE [ GUTEN TAG! ES FREUT MICH, SIE WIEDER EINMAL ZU SEHEN. ]
ENDE

PR UNFREUNDLICHE.BEGRUESSUNG
 DRUCKE [ WAS WOLLEN SIE DENN SCHON WIEDER HIER? ]
ENDE
```

Auch hier sollen die Punkte in den Prozedurnamen der besseren Lesbarkeit dienen.

Lassen Sie uns unsere Begrüßungsprozeduren noch etwas ausbauen. Dazu ein kleiner Exkurs. Beim Aufruf ZUFALLSZAHL 100 (kurz: ZZ 100) gibt Logo einen willkürlichen ganzzahligen Wert zwischen 0 und 99 (je einschließlich) zurück. Allgemein gibt ZUFALLSZAHL :N einen ganzzahligen Zufallswert zwischen 0 und :N-1 (jeweils einschließlich) zurück.

Probieren Sie: WH 100 [DZ ZZ 20]

(DRUCKEZEILE, kurz DZ , bewirkt, daß der Blinker nach dem Ausdrucken auf den linken Rand der nächsten Zeile vorrückt, während er bei DRUCKE,kurz DR, genau hinter dem ausgedruckten Text stehenbleibt).

Mit Hilfe von ZUFALLSZAHL können wir jetzt die folgende Begrüßungsprozedur schreiben:

```
PR UNBERECHENBARE.BEGRUESSUNG
 WENN (ZUFALLSZAHL 2) = 0 DANN  UNFREUNDLICHE.BEGRUESSUNG ____________
 __________________________ SONST FREUNDLICHE.BEGRUESSUNG
ENDE
```

Hierzu einige Bemerkungen:

(1.) In den meisten Logo-Versionen muß man die zweite und dritte Zeile der obigen Prozedur ohne RETURN fortlaufend über den rechten Bildschirmrand schreiben. Der besseren Lesbarkeit halber wurden oben die Unterstreichungszeichen eingefügt. Sie sollen oben und im folgenden stets bedeuten, daß der Text fortlaufend ohne Berücksichtigung des Bildschirmrandes einzugeben ist. Eine vollgeschriebene Zeile "läuft" beim Weiterschreiben dann nach links in die nächste Zeile über.

(2.) Mit Hilfe der Logo-Grundwörter WENN ... DANN ... SONST kann man die Ausführung von Befehlen von bestimmten Bedingungen abhängig machen. Sie lassen sich im Prinzip beliebig verschachteln. Das heißt, Formulierungen wie

WENN Bedingung1 DANN Handlung1 ______
______ SONST WENN Bedingung2 DANN Handlung2 ______
______ SONST ...

sind möglich.

Das Grundwort DANN ist nur "syntaktischer Zucker", man kann grundsätzlich darauf verzichten. An Stelle des obigen Beispiels könnte man also auch schreiben:

WENN Bedingung1 Handlung1 ______
______ SONST WENN Bedingung2 Handlung2 ______
______ SONST ...

Die Fallunterscheidung WENN ... DANN ... SONST ... wird in Kapitel 5 über die Kontrollstrukturen von Logo (besonders Abschnitt 5.2) noch in vertiefter Form behandelt.

Aufgabe 2.12: Schreiben Sie die folgenden Prozeduren:
UEBERWIEGEND.FREUNDLICHE.BEGRUESSUNG und
UEBERWIEGEND.UNFREUNDLICHE.BEGRUESSUNG

2.4 Funktionen

2.4.1 Der Unterschied zwischen Funktionen und Prozeduren

Lassen Sie uns mit einem Beispiel beginnen. Die meisten Logo-Versionen verfügen über relativ wenige numerische Grundfunktionen. Zum Beispiel gibt es oft keine Funktion QUADRAT, welche das Quadrat der eingegebenen Zahl zurückgibt; etwa so:

QUADRAT 5
ERGEBNIS: 25

(QUADRAT hat in diesem Abschnitt also eine andere Bedeutung als im vorigen, wo die Prozedur dieses Namens ein Quadrat zeichnen sollte).

Das Fehlen der QUADRAT-Funktion ist nicht besonders tragisch, denn an Stelle von QUADRAT :A könnte man stets :A * :A schreiben. Dennoch sei hier dieses Beispiel benutzt, weil es einfach zu verstehen und typisch für viele ähnliche Situationen ist.

Eine schlechte Realisierung der Funktion QUADRAT wäre:

```
PR QUADRAT.SCHLECHT  :X
 DRUCKE  :X * :X
ENDE
```

Warum ist diese Lösung nicht gut? Nun, wir können zwar eingeben:

QUADRAT.SCHLECHT 5
und Logo reagiert mit:
25

Im Gegensatz etwa zur Sinus-, Cosinus- und Quadratwurzelfunktion (SIN, COS und QW) können wir die Prozedur QUADRAT.SCHLECHT nicht mit anderen Funktionen verknüpfen. Während man zum Beispiel ohne weiteres

SIN COS 60 (lies Sinus von Cosinus von 60 Grad), oder
COS QW 50 oder
SIN COS SIN QW QW QW 500

von Logo berechnen lassen kann, könnte man nicht

QW QUADRAT.SCHLECHT 80 oder
QUADRAT.SCHLECHT QUADRAT.SCHLECHT 3 oder allgemein
F QUADRAT.SCHLECHT 4 (lies: F von QUADRAT.SCHLECHT von 4)

auswerten lassen, wo mit F eine benutzerdefinierte Funktion gemeint ist. Der Versuch, das zu tun, hat nur die Fehlermeldung:

KEINE RUECKGABE VON QUADRAT.SCHLECHT

zur Folge. Denn der Befehl DRUCKE "frißt" den berechneten Wert :X * :X auf. Er steht dann zwar auf dem Bildschirm, kann aber nicht mehr in anderen Funktionen verwendet werden.

Eine bessere Lösung stellt die folgende Prozedur dar:

```
PR QUADRAT  :X
 RUECKGABE  :X * :X
ENDE
```

Der Befehl RUECKGABE (kurz: RG) bewirkt, daß der Funktionswert an die Stelle zurückgegeben wird, von der aus die Funktion QUADRAT aufgerufen wurde. Die Verwendung des RUECKGABE-Befehls hat zur Folge, daß mit dem ermittelten Funktionswert (= Ergebnis) weitergerechnet werden kann.

Man kann sich Funktionen im Sinne des nachfolgenden Bildes wie Maschinen vorstellen, in die man etwas hineinsteckt und die eine bestimmte Ausgabe produzieren, die wieder in eine andere Funktion eingespeist werden kann.

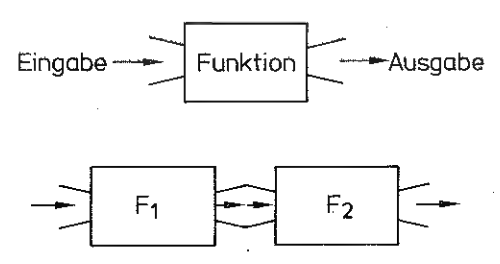


Abbildung 2.4: Funktion und Funktionsverkettung

Im Beispiel SIN QUADRAT 4 wird zunächst QUADRAT 4 berechnet und dieser Wert (also: 16) dann an die Funktion SIN weitergereicht. SIN 16 hätte zu demselben Ergebnis geführt.

Experimentieren Sie:

```
QW QUADRAT 5
QUADRAT QW 6
QUADRAT QUADRAT 3.5
QUADRAT QUADRAT QUADRAT QUADRAT 2
```

Prozeduren, deren Ablauf in der Weitergabe eines Funktionswertes resultiert, werden **Funktionen** genannt. Im Editor werden sie wie sonstige Prozeduren auch durch die Abkürzung PR (und nicht etwa durch eine spezielle Abkürzung, wie zum Beispiel FN oder FKT gekennzeichnet). Funktionen sind also spezielle Prozeduren.

2.4.2 Funktionswerte, die nicht weiterverarbeitet werden

Im Beispiel COS QUADRAT 2.5 wird der Funktionswert 6.25, welchen die Funktion QUADRAT liefert, von der Cosinus-Funktion weiterverarbeitet. Dagegen erfolgt beim Aufruf QUADRAT 3.2 keine Weiterverarbeitung des Funktionswerts. Die verschiedenen Logo-Versionen reagieren im letzteren Fall unterschiedlich.

Im MIT Logo und daraus abgeleiteten Versionen (wie Terrapin, Krell, IWT oder Commodore Logo) wird davon ausgegangen, daß der Benutzer eben daran interessiert ist, diesen Funktionswert zu sehen. Also wird das Wort ERGEBNIS: und dann der Funktionswert ausgedruckt.

In den LCSI-Versionen von Logo (also den Versionen der Firma Logo Computer Systems Inc., Montreal, von der das offizielle Apple Logo, IBM Logo, Macintosh Logo stammen) wird dagegen davon ausgegangen, daß nicht voll verarbeitete Funktionswerte gar nicht vorkommen dürfen. Wenn man in diesen Logo-Versionen zum Beispiel QUADRAT 3 eintippt, so erscheint die Fehlermeldung der Art:

YOU DIDN'T TELL ME WHAT TO DO WITH 9 oder:
YOU DON'T SAY WHAT TO DO WITH 9 oder:
I DON'T KNOW WHAT TO DO WITH 9

Das Mindeste, was man in diesen Logo-Versionen mit einem Funktionswert tun muß, um seine Ermittlung ordentlich abzuschließen ist, ihn auszudrucken. Um die obige Fehlermeldung zu vermeiden muß man also zum Beispiel PRINT QUADRAT 3 eingeben. (Natürlich hätten in der Prozedur QUADRAT in einer englischsprachigen Logo-Version auch englische Grundwörter verwendet werden müssen). Nach der Befehlsfolge PRINT QUADRAT 3 druckt LCSI Logo einfach die Zahl 9 aus.

Auch im MIT Logo kann man selbstverständlich DRUCKE QUADRAT 3 eingeben. In diesem Fall reagiert das MIT Logo genau wie das LCSI Logo, das heißt, das Wort ERGEBNIS: entfällt und es erscheint nur die 9 auf dem Bildschirm.

Lassen Sie mich an dieser Stelle vorab darauf hinweisen, daß genau dies ein gutes Beispiel dafür ist, wie leicht die "Oberfläche" von Logo verändert und speziellen

Benutzerbedürfnissen angepaßt werden kann. Wenn Sie nämlich etwa ein LCSI Logo gekauft haben und Ihnen im Laufe der Zeit die Fehlermeldung "YOU DIDN'T TELL ME WHAT TO DO WITH ...' auf die Nerven geht, können Sie im Rahmen von Logo leicht das Verhalten von Logo ändern und zum Beipiel eine Reaktion wie im MIT Logo erzwingen. Ein Beispiel hierfür wird in Abschnitt 5.7 gegeben.

Neben dem Ausdrucken gibt es natürlich auch noch andere Möglichkeiten, einen Funktionswert "aufzufressen". So wurde etwa bereits in Kapitel 1 im Beispiel AUFKURS RICHTUNG 100 10 der Funktionswert der Funktion RICHTUNG, nämlich der Wert 30.9632, im Befehl AUFKURS weiterverarbeitet.

Diese Bemerkungen beziehen sich allerdings nur auf Funktionsaufrufe im Direktausführungsmodus (englisch: top level mode). Erfolgt ein Funktionsaufruf von einer Prozedur aus und wird der Funktionswert nicht weiterverarbeitet, so kommt stets (also auch bei den MIT-Versionen von Logo) eine Fehlermeldung. Probieren Sie dies mit den beiden folgenden Funktionen aus:

```
PR FOO
 RUECKGABE 2
END

PR BAR
 FOO
 DRUCKE 3
END
```

Der Aufruf FOO liefert wie bisher den Wert 2, aber der Aufruf BAR führt zu der Fehlermeldung: WAS SOLL MIT 2 GESCHEHEN IN BAR?

(FOO, BAR und BAZ sind beliebte Funktionsnamen in Logo, besonders für Funktionen, die man nur zu Test- oder Demonstrationszwecken benötigt).

2.4.3 RUECKGABE als Kontrollstruktur

Der Befehl RUECKGABE hat außer der Weitergabe des Funktionswertes noch eine zweite Wirkung: er bewirkt auch stets, daß die Abarbeitung der Funktion mit der Wertrückgabe abgeschlossen wird. Das heißt, RUECKGABE spielt auch noch die Rolle einer Kontrollstruktur. Ein Beispiel:

```
PR ABSOLUTBETRAG  :X
 WENN :X < 0 DANN RUECKGABE (-:X)
 RUECKGABE :X
ENDE
```

Dieses Thema wird in Kapitel 5 noch ausführlicher behandelt.

2.4.4 Weitere Beispiele von Funktionen

2.4.4.1 Die Abstands-Funktion

Die QUADRAT-Funktion geht als Baustein in eine Fülle von Funktionen ein. So zum Beispiel in die Funktion ABSTAND :X1 :Y1 :X2 :Y2 , welche den Abstand des Punktes P mit den Koordinaten (:X1 / :Y1) vom Punkt Q mit den Koordinaten (:X2 / :Y2) als Funktionswert zurückgibt.

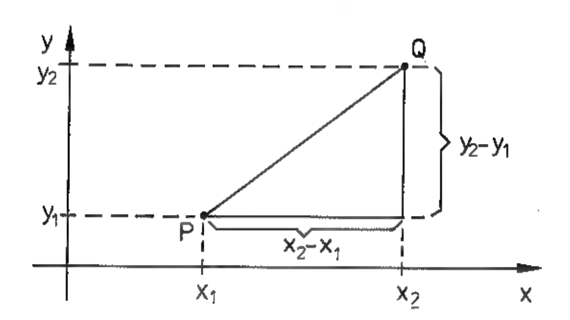


Abbildung 2.5: Abstand zweier Punkte

In diese Funktion fließt der **Satz des Pythagoras** ein. Sie kann im wesentlichen nur folgendermaßen aussehen:

```
PR ABSTAND  :X1  :Y1  :X2  :Y2
 RUECKGABE QW ((QUADRAT ( :X2 - :X1 )) + (QUADRAT ( :Y2 - :Y1 )))
ENDE
```

Wie Sie sehen, wurde reichlich Gebrauch von runden Klammern gemacht. Sie dienen zweierlei Zielen:

(1.) Durch die **Klammern** soll eine bestimmte Auswertungsreihenfolge erzwungen werden. Normalerweise wertet Logo die eingegebenen Ausdrücke "von rechts nach links" aus. Zum Beispiel würde

QUADRAT 2 + QUADRAT 3

das Ergebnis 121 liefern, weil Logo den ungeklammerten Term folgendermaßen auswertet:

- zuerst wird QUADRAT 3 mit dem Ergebnis 9 ausgewertet;
- danach wird 2 + 9 ausgewertet, Ergebnis: 11;
- und schließlich wird QUADRAT 11 berechnet, also 121.

(2.) Darüberhinaus dienen Klammern oft der optischen Gliederung von Daten ohne Auswirkung auf das Ergebnis des Auswertung. So sind zum Beispiel die Klammern

um den Teilterm :X2 - :X1 im Hinblick auf die Auswertung ohne Belang. Klammern dienen in Logo häufig derartigen Giederungsbedürfnissen. Dabei kommt es häufig vor, daß man eine Funktion mit ihren Argumenten als eine Gruppierung zum Ausdruck bringen will. Man schreibt dann zum Beispiel an Stelle von F :X :Y :Z auch (F :X :Y :Z); gelesen: F von :X, :Y und :Z.

2.4.4.2 Widerstand bei Parallelschaltung

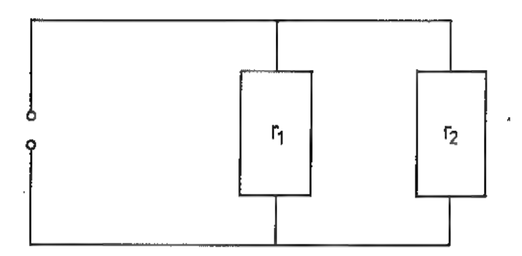


Abbildung 2.6: Parallelschaltung

Die Abbildung zeigt einen Stromkreis, in dem zwei Widerstände parallel geschaltet sind. Haben die Einzelwiderstände die Werte r1 und r2 (in Ohm), so gilt nach den **Kirchhoffschen Gesetzen** für den Gesamtwiderstand r :

$1/r = 1/r1 + 1/r2$ beziehungsweise $r = r1 * r2 / (r1 + r2)$

Die Prozedur zur Berechnung des Gesamtwiderstands aus den Einzelwiderständen soll kurz WP heißen (für Widerstand bei Parallelschaltung).

```
PR WP  :R1 :R2
 RUECKGABE :R1 * :R2 / ( :R1 + :R2 )
ENDE
```

Betrachten wir nun den folgenden Stromkreis:

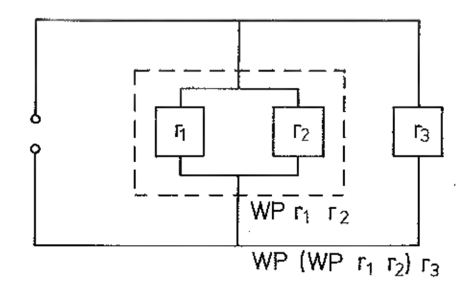


Abbildung 2.7: Verschachtelte Parallelschaltung

Der Stromkreis läßt sich deuten als die Parallelschaltung der beiden Widerstände r und r3, wobei sich r wiederum aus der Parallelschaltung von r1 und r2 ergibt. Also ist der Gesamtwiderstand:

WP (WP :R1 :R2) :R3 (2.4)

erstes Argument von WP	zweites Argument von WP

Man hätte den obigen Ausdruck auch ohne Klammern schreiben können, ohne den Auswertungsvorgang und damit das Ergebnis zu verändern:

WP WP :R1 :R2 :R3 (2.5)

Da die Funktion WP genau zwei Argumente erwartet, werden die Argumente in (2.5) von Logo automatisch richtig gruppiert: Zuerst wird WP :R1 :R2 berechnet (mit Ergebnis :R) und dann wird WP mit den Argumenten :R und :R3 ausgewertet.

2.4.4.3 Funktionen zur Konversion von Winkelmaßen

Die Logo-Grundfunktionen SIN und COS verlangen als Eingabe jeweils einen Winkel im Gradmaß. Häufig hat man es aber mit Winkeln im Bogenmaß zu tun, auf die man die trigonomischen Funktionen anwenden möchte. Es wäre also günstig, wenn wir eine Funktion hätten, die einen im Bogenmaß gegebenen Winkel ins Gradmaß überführt. Die folgende Funktion tut dies:

```
PR GRADMASS  :X
 RUECKGABE  :X * 180 / 3.14159
ENDE
```

Testen Sie:

GRADMASS 0.5
ERGEBNIS: 28.6479

GRADMASS 3.14159
ERGEBNIS: 180.0

GRADMASS 0.7854
ERGEBNIS: 45.0001

SIN GRADMASS 0.7854
ERGEBNIS: 0.707108

SIN 45
ERGEBNIS: 0.707107

GRADMASS 1.0472
ERGEBNIS: 60.0002

SIN GRADMASS 1.0472
ERGEBNIS: 0.866026

SIN 60
ERGEBNIS: 0.866025

Wenn wir die Sinusfunktion fast ausschließlich auf Winkel im Bogenmaß anwenden wollen, können wir uns auch eine neue Sinusfunktion definieren:

```
PR SINUS  :X
 RUECKGABE  SIN  GRADMASS  :X
ENDE
```

Was dem Gradmaß recht ist, sollte dem Bogenmaß billig sein.

Aufgabe 2.13: Schreiben Sie eine Funktion BOGENMASS, die einen im Gradmaß gegebenen Winkel ins Bogenmaß überführt. Testen Sie:

BOGENMASS 45
BOGENMASS 60

BOGENMASS GRADMASS 0.75
GRADMASS BOGENMASS 78

SIN 25
SINUS BOGENMASS 25

2.4.5 Weitere Übungen zum funktionalen Programmieren

Aufgabe 2.14: Schreiben Sie Funktionen PROZENTSATZ, GRUNDWERT, PROZENTWERT. Überlegen Sie, welche Eingabeparameter diese Funktionen jeweils haben müssen.

Aufgabe 2.15: Schreiben Sie eine Funktion MEHRWERTSTEUER, welche auf die Funktion PROZENTWERT zurückgreift.

Aufgabe 2.16: Schreiben Sie Funktionen FAHRENHEIT :C beziehungsweise CELSIUS :F zur Umrechnung der Celsius- in die Fahrenheit-Skala und umgekehrt.

Aufgabe 2.17: Schreiben Sie die Funktionen
ARITHMETISCHES.MITTEL :A :B
GEOMETRISCHES.MITTEL :A :B
HARMONISCHES.MITTEL :A :B

Aufgabe 2.18: Schreiben Sie die Funktionen
LAENGE.DER.DIAGONALE.IM.RECHTECK :A :B
LAENGE.DER.RAUMDIAGONALE.IM.QUADER :A :B :C

Aufgabe 2.19: Schreiben Sie die Funktionen
MINIMUM :A :B
MAXIMUM :X :Y

Aufgabe 2.20: Schreiben Sie eine Funktion SIGNUM ("Vorzeichen") mit den folgenden Eigenschaften:

SIGNUM :X = 1, falls :X > 0
SIGNUM :X = 0, falls :X = 0
SIGNUM :X = -1, falls :X < 0

Aufgabe 2.21: Schreiben Sie Funktionen

LINEARE.FUNKTION :M :B :X
QUADRATISCHE.FUNKTION :A :B :C :X
ANTIPROPORTIONALE.FUNKTION :A :X
INVERS :X

zur Auswertung der folgenden Funktionen (in mathematischer Schreibweise):

f1(x) = m*x + b
f2(2) = a*x*x + b*x + c
f3(x) = a / x
f4(x) = 1 / x

Aufgabe 2.22: (a) Schreiben Sie die Funktion TAN :W und COT :W, die als Eingabe einen Winkel im Gradmaß erwarten.

(b) Schreiben Sie (in Anlehnung an SINUS in Abschnitt 2.4.4.3) die Funktionen COSINUS, TANGENS, COTANGENS, die als Eingabe einen Winkel im Bogenmaß erwarten.

Aufgabe 2.23: Schreiben Sie die (konstanten) Funktionen NULL, EINS, PI, E (Hinweis: Man kann auf die Verwendung von Konstanten häufig dadurch verzichten, daß man an ihrer Stelle konstante Funktionen verwendet).

Die Verwendung solcher konstanter Funktionen kann zum Beispiel folgendermaßen erfolgen:

Umfang eines Kreises mit Radius 3.8: 2 * PI * 3.8.

Aufgabe 2.24: Schreiben Sie Prozeduren für weitere Grundfunktionen aus Ihrem Arbeitsbereich. Legen Sie sich einen Werkzeugkasten (eine "toolbox") von Funktionen an, die Sie häufig verwenden.

2.5 Erste Beispiele zur Gestaltung der "Benutzeroberfläche" von Logo

Die Existenz von Prozeduren und Funktionen ermöglicht bereits eine erste Anpassung von Logo an den Benutzer. Nehmen wir an, Ihnen gefällt ein bestimmtes Grundwort, zum Beipiel ARCTAN nicht. Ihnen wäre vielleicht ARCTG lieber. Wenn Ihnen dies sehr wichtig ist, können Sie die Funktion ohne weiteres umbenennen:

```
PR ARCTG  :X
 RUECKGABE  ARCTAN  :X
ENDE
```

Oder nehmen wir an, Sie arbeiten mit einer englischsprachigen Logo-Version und möchten Kindern, die noch kein Englisch können, eine kleine Igel-"Mikrowelt" zur

Verfügung stellen, in der sie spielend experimentieren können. Dann haben Sie die Möglichkeit, einige der wichtigsten Igelbefehle zu übersetzen; zum Beispiel:

```
TO VORWAERTS  :X
 FORWARD  :X
END

TO RECHTS  :X
 RIGHT  :X
END
```

und so weiter. In den englischsprachigen Versionen steht TO (also der Infinitiv) an Stelle von PR und OUTPUT (kurz: OP) an Stelle von RUECKGABE. Dies ermöglicht für englischsprachige Benutzer sehr natürliche Ausdrucksweisen. Die Funktion "Vorzeichen ändern" würde zum Beispiel so aussehen:

```
TO CHANGE.SIGN  :X
 OUTPUT  (-:X)
END
```

Dem uneingeschränkten Umdefinieren von Logo-Grundbefehlen sind allerdings (besonders bei den kleineren Rechnern) wegen der Beschränktheit des Speichers Grenzen gesetzt.

Kapitel 3: Grundzüge des Logo-Betriebssystems

Dieses Kapitel ist naturgemäß am stärksten von dem speziellen Computer abhängig, auf dem Sie arbeiten. Es ist primär an den MIT-Versionen von Logo (also MIT, Terrapin, Krell, IWT, Commodore Logo) orientiert. In den LCSI-Versionen von Logo haben Sie ähnliche Grundfunktionen, die allerdings meist etwas anders aufzurufen sind. Bitte konsultieren Sie Ihr spezielles Handbuch für etwaige Detailfragen. Oder probieren Sie manche Dinge einfach aus. Sie werden sehr schnell eine Fertigkeit darin entwickeln, durch kleine Tests Antworten auf Fragen zu bekommen, die Sie sonst erst mühsam im Handbuch nachschlagen müßten.

3.1 Der Logo-Editor

Ein Editor ist ein Hilfsprogramm zur Erstellung von Texten. Wir sind mit dem Prozeß des Editierens bisher mehr oder weniger unbewußt an den folgenden beiden Stellen in Berührung gekommen:

- bei der Benutzung des **Zeilen-Editors** im Direktausführungsmodus;
- bei der Benutzung des **Bildschirm-Editors**, mit dessen Hilfe wir Prozeduren geschrieben haben.

Der Bildschirm-Editor bietet wesentlich mehr Möglichkeiten als der Zeilen-Editor.

Eine Bemerkung zum Begriff der Zeile: es ist zu unterscheiden zwischen dem Begriff der **optischen Zeile** und der **logischen Zeile**. Eine optische Zeile ist genau das, was auf derselben Höhe zwischen linkem und rechtem Bildschirmrand steht. Eine logische Zeile ist das, was zwischen zwei RETURN-Zeichen (dem Zeichen mit ASCII-Wert 13) steht. Überschreitet eine logische Zeile den rechten Bildschirmrand, so wird dies von Logo automatisch durch ein Ausrufezeichen (!) zum Ausdruck gebracht. Das Ausrufezeichen selbst ist nur ein Hinweis und gehört nicht zum Zeileninhalt.

In diesem Buch wird das Unterstreichungszeichen benutzt, um anzudeuten, daß Dinge, die auf mehreren optischen Zeilen stehen, zu derselben logischen Zeile gehören; siehe zum Beispiel die Diskussion der Funktionen zum Thema BEGRUESSUNG in Kapitel 2 (Abschnitt 2.3).

Im folgenden wird der Begriff "Zeile" stets im Sinne von "logischer Zeile" verwendet.

3.1.1 Der Bildschirm-Editor von Logo

Grundfunktionen des Bildschirm-Editors (im folgenden kurz als Editor bezeichnet) sind:

- das Einfügen von Zeichen in einen Text;
- das Löschen von Zeichen aus einem Text;
- das Verschieben des Blinkers im Text;
- das Verschieben des Textes am Bildschirm.

Weiterhin gehören zum Gebrauch des Editors natürlich noch:

- die Befehle zum Aufruf des Editors aus dem Direktausführungsmodus;
- die Befehle zum Verlassen des Editors zum Zwecke der Rückkehr in den Direktausführungsmodus.

3.1.1.1 Aufruf des Editors

Eine Möglichkeit haben wir schon kennengelernt. Mit Hilfe des Befehls LERNE Prozedurnamen können wir Logo veranlassen, in den Editor umzuschalten. Auf dem Bildschirm ist danach der Text PR Prozedurname zu sehen. Weitere Möglichkeiten, dasselbe zu erreichen, sind durch die folgenden Kommandos gegeben:

EDIT Prozedurname
ED Prozedurname
PR Prozedurname

In englischsprachigen Logo-Versionen wird der Editor üblicherweise mit dem Befehl TO Prozedurname aufgerufen.

Will man den zuletzt editierten Text nochmals editieren, so reicht der Befehl EDIT oder kurz ED (ohne Prozedurnamen). Mit dem Befehl EDIT ALLES kann man alles bisher Editierte "auf einen Schlag" editieren.

Bei manchen Logo-Versionen ist es möglich, eine in eckigen Klammern zusammengefaßte Gruppe von Prozeduren zu editieren.

Der editierte Text wird in einem speziellen Speicherbereich, dem sogenannten **Editier-Speicher** oder Editier-Puffer (englisch: edit buffer) abgelegt.

3.1.1.2 Verlassen des Editors

Mit Ctrl-C wird bewirkt, daß zugleich mit dem Ausstieg aus dem Editor die editierten Prozeduren "gelernt" werden. Beim Commodore 64 bewirkt das Drücken der RUN/STOP-Taste dasselbe wie Ctrl-C.

Nach Ctrl-G wird der Editor verlassen, ohne daß das Editierte gelernt wird. Der editierte Text befindet sich aber auch nach Ctrl-G nach wie vor im Editier-Speicher und kann mit EDIT wieder verfügbar gemacht werden.

Eine nur in sehr seltenen Fällen zu empfehlende Art, den Editiermodus zu verlassen, ist das Ausschalten des Computers.

3.1.1.3 Einfügen von Text

Im Logo-Editor sind Sie stets im Einfüge-Modus (englisch: insert-mode). Das heißt, wenn Sie ein Zeichen tippen, wird dies an die Stelle geschrieben, auf der der Blinker steht und der gesamte Text rechts vom Blinker wird verschoben.

Mit Hilfe von Ctrl-O können Sie eine neue Zeile "öffnen". Dadurch läßt sich unter anderem das optisch möglicherweise etwas irritierende fortlaufende Weiterschieben des Textes rechts vom Blinker vermeiden.

3.1.1.4 Löschen von Text

ESC: Durch Tippen der "Escape"-Taste wird das Zeichen links vom Blinker gelöscht;
DEL: wie ESC
Ctrl-D: durch diese Taste wird das Zeichen unter dem Blinker gelöscht;

Der rechts vom gelöschten Zeichen stehende Text wird entsprechend nach links verschoben.

Ctrl-X: alle Zeichen der Zeile rechts vom Blinker werden gelöscht;
Ctrl-K: bei einigen Versionen an Stelle von Ctrl-X;

(Ctrl-K und Ctrl-X sind sehr stark von der speziellen Logo-Version abhängig).

Ctrl-Y ("yank"): kopiert bei manchen Logo-Versionen den zuletzt gelöschten Text an die Stelle, wo der Cursor gerade steht.

3.1.1.5 Blinker-Bewegung (ohne zu löschen)

Pfeiltasten: jeweils in die angegebene Richtung;
Ctrl-P: eine Zeile nach oben (das "P" erinnert an: "previous line");
Ctrl-N: eine Zeile nach unten (das "N" erinnert an: "next line");
Ctrl-A: auf den Anfang der Zeile;
Ctrl-E: auf das Ende der Zeile;

Auch beim Verschieben des Textes (siehe unten) wird der Blinker so positioniert, daß er im Bildfenster bleibt.

3.1.1.6 Verschieben des Textes ("Rollen")

Ctrl-B: der dargestellte Text wird um eine Bildschirmseite rückwärts verschoben (das "B" erinnert an: "backward");
Ctrl-F: der dargestellte Text wird um eine Bildschirmseite vorwärts verschoben (das "F" erinnert an "forward");
Ctrl-L: der Text wird so gerollt, daß der Blinker etwa in der Mitte der Bildschirmseite steht.

Auch wenn der Blinker bei einer Blinkerbewegung den Bildschirmrand überschreiten würde, wird der Text so gerollt, daß der Blinker im Bildfenster bleibt.

3.1.2 Der Zeilen-Editor

Die meisten Befehle des Bildschirm-Editors können auch dazu benutzt werden, um im Zeilen-Editor des Direktausführungsmodus Tippfehler zu korrigieren. Natürlich gilt dies nur für Befehle, die nicht dazu führen, daß der Blinker seine gegenwärtige Zeile verläßt.

Die Eingabe von Ctrl-P hat im Zeilen-Editor eine neue Bedeutung: sie bewirkt (meistens), daß die zuletzt eingegebene Zeile wiederholt wird. Dadurch kann man sich das wiederholte Tippen langer Zeilen ersparen.

Ein Beispiel: nehmen wir an, Sie haben die Befehlsfolge

AUFKURS RICHTUNG QW XKO + 80 COS YKO + 50

eingegeben und ausführen lassen. Als nächstes wollen Sie es mit 70 an Stelle von 50 probieren. Dann können Sie folgendermaßen vorgehen:

Ctrl-P

Auf dem Bildschirm erscheint wieder:

AUFKURS RICHTUNG QW XKO + 80 COS YKO + 50

Der Blinker steht rechts neben der 50.

Zweimaliges Tippen der DEL-Taste (bzw. ESC-Taste) löscht die 50. Nun können Sie stattdessen 70 eingeben und die neue Zeile mit RETURN zur Ausführung bringen.

Im LCSI Logo ist Ctrl-R (für "repeat") an Stelle von Ctrl-P zu verwenden.

3.2 Der Arbeitsspeicher

Die definierten Prozeduren, Funktionen und Variablen werden im Arbeitsspeicher abgelegt. Einige Logo-Grundbefehle dienen der Verwaltung des Arbeitsspeichers.

ZEIGE (kurz ZG) Prozedurnamen; zum Beispiel: ZEIGE QUADRAT. Die angegebene Prozedur wird aufgelistet. Sie steht dann am Bildschirm beziehungsweise (falls ein Drucker angeschlossen ist) auf Papier; kann aber in dieser Form nicht editiert werden. Außerdem kann man nach ZEIGE auch ALLES oder PROZEDUREN eingeben.

ZEIGE TITEL (kurz ZT) listet alle vom Benutzer definierten Prozeduren und Funktionen mit den jeweiligen formalen Parametern auf.

Ctrl-W dient (im Wechsel mit irgendeiner anderen Taste) in der Art eines Wechselschalters dazu, die Ausführung eines Programmes zu unterbrechen und wieder fortzusetzen. Auf diese Weise kann man den ZEIGE-Vorgang (zum Beispiel nach ZEIGE ALLES) kurz unterbrechen, um sich bestimmte Prozeduren genauer anzusehen.

VERGISS (kurz VG) Prozedurnamen
Beispiel: VERGISS ROSETTE
Die genannte Prozedur wird gelöscht.

Weitere später zu diskutierende Befehle sind: ZEIGE NAMEN, VERGISSNAME (kurz VN).

Der Arbeitsspeicher wird von Logo **dynamisch** verwaltet. Das heißt, daß zunächst beanspruchter, dann aber wieder freigewordener Speicherplatz dem Logo-Arbeitsspeicher wieder ordnungsgemäß zur Verfügung gestellt wird. Diesen Prozeß nennt man **garbage collection**. Wenn der Speicherplatz knapp zu werden droht, führt Logo automatisch eine garbage collection durch. Eine garbage collec-

tion kann auch vom Benutzer durch den Befehl .GCOLL erzwungen werden. Jede garbage collection benötigt eine gewisse Zeit. Vor besonders "zeitkritischen" Routinen kann sich deshalb eine erzwungene garbage collection als nützlich erweisen, damit sie sich nicht zu einem unerwünschten Augenblick einstellt. Der Anfänger hat aber mit dem Problem der garbage collection praktisch nichts zu tun.

Der Befehl .KNOTEN gibt den noch freien Arbeitsspeicher, gemessen in "Knoten" wieder. (Ein Knoten entspricht 5 Bytes).

ADE
Dieser Befehl löscht den Arbeitsspeicher und startet Logo neu.

3.3 Das Arbeiten mit Disketten

Man möchte einmal erstellte Prozeduren und Funktionen natürlich konservieren. Diesem Ziel dienen die Befehle BEWAHRE (kurz: BW) und LADE.

Ein Beispiel: der Befehl BEWAHRE "FIGUREN hat zur Folge, daß der gesamte Inhalt des Arbeitsspeichers als eine Datei unter dem Namen FIGUREN auf Diskette geschrieben wird. Wenn man diese Prozeduren später einmal wieder braucht, dann kann man sie mit dem Befehl LADE "FIGUREN wieder in den Arbeitsspeicher zurückholen.

Man kann, solange der Speicher reicht, zu jedem Zeitpunkt beliebig viele früher abgespeicherte Dateien in den Arbeitsspeicher laden. Ihr Inhalt wird dem bisherigen Inhalt des Arbeitsspeichers hinzugefügt. Bei Gleichnamigkeit werden allerdings die alten Prozeduren durch die neu geladenen Prozeduren überschrieben.

Reicht der Arbeitsspeicher des Computers nicht aus, um eine zu ladende Datei aufzunehmen, so bringt Logo die Fehlermeldung:

SPEICHER VOLL!

Weitere Befehle für das Arbeiten mit Diskette:

INHALT (kurz IH) gibt das Inhaltsverzeichnis (englisch: catalog oder directory) der Diskette wieder.

VERGISSDATEI (kurz VD)

Beispiel:
VERGISSDATEI "FIGUREN
löscht die Datei des Namens FIGUREN von der Diskette.

Die Diskettenbefehle BEWAHREBILD und VERGISSBILD dienen dem Aufzeichnen und Löschen von Graphiken.

Man beachte: Im MIT Logo ist bei EDIT und ZEIGE der nachfolgende Name nicht mit einem vorangestellten Anführungszeichen (") zu versehen (kurz: zu quoten). Bei den Diskettenbefehlen (BEWAHRE, LADE, VERGISSDATEI ...) muß er dagegen gequotet werden. LCSI Logo ist hier konsequenter: in dieser Logo-Version muß der Name stets gequotet werden (falls er nicht in der Form eines Funktions- oder Variablenwertes zur Verfügung gestellt wird).

Kapitel 4: Die fundamentalen Datentypen von Logo

4.1 Zahlen

Logo verfügt über zwei Arten von Zahlen: einen Ausschnitt aus dem Bereich der **ganzen Zahlen** und einen Ausschnitt aus dem Bereich der **Dezimalzahlen**. Dezimalzahlen werden im Computer als sogenannte **Gleitkommazahlen** realisiert. Wie groß der Bereich der von Logo aus zugänglichen Zahlen ist, hängt von dem speziellen System ab, mit dem Sie arbeiten.

Das MIT Logo kann ganze Zahlen zwischen -2147483648 und +2147483648 (also -2^{31} und 2^{31}) verarbeiten. Die zugänglichen positiven Dezimalzahlen liegen zwischen 10^{-38} und 10^{38}.

Wenn Logo mit ganzen Zahlen rechnet und weder die Ausgangszahlen noch die Zwischenergebnisse noch das Endergebnis den oben angegebenen ganzzahligen Bereich überschreiten, dann ist das Ergebnis numerisch korrekt.

Rechnungen mit Gleitkommazahlen führen meist nur zu näherungsweise richtigen Ergebnissen. Denn in jedem Intervall auf der Zahlengeraden gibt es unendlich viele Dezimalzahlen. Der Speicher des Computers ist jedoch endlich. Also muß jede Dezimalzahl durch eine geeignete Gleitkommazahl angenähert werden. Im MIT Logo verfügen die Gleitkommazahlen nur über maximal 7 wesentliche Ziffern.

Eine weitere Ungenauigkeit entsteht dadurch, daß Dezimalzahlen intern in die **Binärdarstellung** (Darstellung im Zweiersystem) umgesetzt und nur beim Ausdrucken wieder als Dezimalzahlen dargestellt werden. Auch diese Umwandlungsprozesse vom Zehnersystem ins Zweiersystem und zurück führen zu Genauigkeitsverlusten. Hier eines von vielen unschönen Beispielen im Bereiche der Gleitkommaarithmetik:

```
1 / 3
ERGEBNIS: 0.333333

0.333333 = 1 / 3
ERGEBNIS: FALSCH
```

Im allgemeinen hat der Benutzer wenig mit dieser automatischen Umwandlung von Dezimalzahlen in Binärzahlen zu tun. Auch die gelegentlich auftretenden sehr kleinen **Rundungsfehler** stören meist nicht, da sie das Ergebnis schlimmstenfalls um wenige Promille verfälschen. Problematisch wird der Sachverhalt jedoch dann, wenn sich viele kleine Fehler akkumulieren und sich so lawinenartig fortpflanzen.

In einigen Fällen, so zum Beispiel bei Teilbarkeitsproblemen, sind absolut korrekte Ergebnisse unerläßlich. Man muß dann sicherstellen, daß sich die Rechnungen nur im zulässigen ganzzahligen Bereiche bewegen. Dies ist immer dann der Fall, wenn:

- Die Ausgangszahlen als ganze Zahlen eingegeben werden.
  Beispiele für ganze Zahlen:

3
17
-128
Dagegen werden die folgenden Zahlen als Gleitkommazahlen behandelt:
4.00
5.
12E20
123456789012345 (zu groß für das Ganzzahlformat)

- Weder die Ausgangszahlen noch die Zwischenergebnisse noch das Endergebnis den Bereich der in Logo darstellbaren ganzen Zahlen überschreiten.
- Nur Rechenoperationen und Funktionen verwendet werden, die wieder zu ganzen Zahlen im zulässigen Bereiche führen. Dies sind:

  +, *, -, DIV, REST, INT, RUNDE, ZUFALLSZAHL

  sowie Funktionen, die nur diese Operationen benutzen.

Ein charakteristisches Merkmal von Logo ist, daß es Datentypen, also zum Beispiel "ganze Zahl" oder "Gleitkommazahl", erkennt und arithmetische Operationen so durchführt, daß sie dem Datentyp entsprechen. Probieren Sie folgendes aus:

2 + 3
ERGEBNIS: 5

2. + 3 oder 2 + 3. oder 2.0 + 3 oder 2 + 3.0 oder 2.0 + 3.0
ERGEBNIS: 5.0 (beziehungsweise 5.)

Solange die Eingaben alle ganzzahlig sind, wendet Logo die Ganzzahl-Addition an; sobald mindestens eine Eingabe nicht ganzzahlig ist, arbeitet Logo mit der Gleitkomma-Addition, obwohl wir jedesmal dasselbe Additionszeichen verwendet haben.

Operationen mit der Gleitkomma-Division führen stets zu Gleitkommazahlen; auch wenn das Ergebnis vom mathematischen Standpunkt eine ganze Zahl ist. Beispiel:

6/2
ERGEBNIS: 3.0

Für Gleitkommazahlen gibt es verschiedene Darstellungsformate. Hier einige Beispiele:
2.0
3.14
2.71E3 (E3 bedeutet: mal 10^3; E steht für Exponent)
5.736N2 (N2 bedeutet: mal 10^{-2}; N steht für negativer Exponent)

Es gilt also (theoretisch):

2.71E3 ist wertgleich mit 2710.0;
5.736N2 ist wertgleich mit 0.05736.

Aber beachten Sie die merkwürdigen Ergebnisse des folgenden Dialogs:

2.71E3
ERGEBNIS: 2710

2.71E3 = 2710
ERGEBNIS: FALSCH

2.71E3 = 2710.0
ERGEBNIS: FALSCH

2710.0 = 2710
ERGEBNIS: WAHR

Diese unglaublich schlechten Ergebnisse im Bereiche der Gleitkommaarithmetik sind allerdings nicht auf Logo beschränkt; man findet sie praktisch in jeder Programmiersprache. Wegen der geringeren Interaktivität sind diese Mängel in anderen Sprachen jedoch meist nicht so leicht aufzudecken.

Die soeben beschriebenen Beispiele beziehen sich auf den Apple II Computer. Da die Rechnerartihmetik stark geräteabhängig ist, können die Dialogbeispiele bei anderen Computern zu unterschiedlichen Ergebnissen führen.

4.2 Wörter

Ein **Wort** ist eine Zeichenkette, die aus Buchstaben, Ziffern und speziellen Sonderzeichen bestehen kann. Beispiele für Wörter:

XYZ
CAMBRIDGE
HURRA
DONAUDAMPFSCHIFFAHRTSKAPITAEN
X3P5
127
3.472
T.L.C.
A$$B##C17

Wie Sie sehen, werden die Zahlen auch zu den Wörtern gezählt. Mit den Sonderzeichen sollte man, wie schon bei den Prozedurnamen erwähnt, vorsichtig sein. Vor allem für den Anfänger empfiehlt es sich, alle Arten von Klammern, Anführungszeichen und das Leerzeichen in Wörtern zu vermeiden. Dies sollte nicht allzu schwierig sein.

Bis auf eine einzige, weiter unten im Abschnitt über die Einbettung von Leerzeichen in Wörter zu beschreibende Ausnahme, ist **das Ende eines jeden Wortes mit dem Leerzeichen (englisch: space oder blank) gegeben.**

In der folgenden Zeile stehen also zum Beispiel fünf Wörter:
SHERLOCK HOLMES FAENGT DEN DIEB.

4.2.1 Wörter im Prozeß der Verarbeitung durch Logo

Im Direktausführungsmodus versucht Logo, der Reihe nach alles zu verarbeiten, was ihm "vorgesetzt" wird. Man sagt auch, Logo interpretiert die eingelesenen

Befehle. Zur Interpretation gehört erstens das richtige Erkennen des jeweiligen Befehls (mit eventueller Fehlermeldung) und zweitens die Ausführung des Befehls, wenn er als korrekt erkannt wurde.

Wird zum Beispiel SIN 30*2 eingegeben, so erkennt Logo, daß mit dem ersten Wort die Sinusfunktion (aus seinem Grundwortschatz) gemeint ist, daß 30 eine Zahl, * das Multiplikationszeichen und 2 ebenfalls eine Zahl ist. Logo berechnet zunächst 30*2, wendet auf diesen Wert die Sinusfunktion an, gibt das Ergebnis zurück, druckt als Bereitschaftszeichen (englisch: prompt) ein Fragezeichen und wartet auf weitere Eingaben.

Wenn wir jetzt zum Beispiel ROSETTE 50 eingeben, so sucht Logo das Wort ROSETTE zunächst unter seinen Grundwörtern (englisch: primitives). Nachdem Logo festgestellt hat, daß ROSETTE kein Grundwort ist, schaut es im Arbeitsspeicher nach, ob ROSETTE eine vom Benutzer definierte Prozedur ist. Wenn es ROSETTE dort findet, führt es die Prozedur aus, andernfalls bringt Logo die Fehlermeldung "PROZEDUR ROSETTE UNBEKANNT".

Logo versucht also, jedes eingegebene Wort zu interpretieren und seinen Auswertungsmechanismus auf dieses Wort anzuwenden. Wenn wir etwa
DRUCKE REUTLINGEN
eingeben, so wird nicht etwa das Wort REUTLINGEN ausgedruckt, sondern Logo versucht, REUTLINGEN unter den Prozedurnamen zu finden und diese Prozedur auszuführen. Wenn dies nicht gelingt, druckt es eine entsprechende Fehlermeldung.

Nun möchte man aber gelegentlich auch einfachen Text ausdrucken oder auf andere Art verarbeiten. Für diesen Fall verfügt Logo über eine Operation, welche die übliche Interpretation von Wörtern unterbindet: das Anführungszeichen " (englisch: **quote**). (Beim MIT Logo ist speziell das aus zwei Strichen bestehende Anführungszeichen gemeint; nicht etwa das aus einem Strich bestehende Anführungszeichen). Ein Wort, dem ein Anführungszeichen vorangestellt ist, wird von Logo ohne weitere Auswertung zurückgegeben. Man sagt dann, dieses Wort sei gequotet. (Für Mathematiker: das Anführungszeichen arbeitet wie die identische Funktion). Einige Beispiele:

"XYZ
ERGEBNIS: XYZ

DRUCKE "REUTLINGEN
REUTLINGEN

"2*3
ERGEBNIS: 2*3

Es ist wichtig, sich klar zu machen, daß das Anführungszeichen eine Logo-Funktion ist. Deswegen steht es auch nur am Anfang und nicht am Ende des jeweiligen Wortes. (Das Ende eines Wortes ist durch das Leerzeichen gegeben). Im Unterschied zu den übrigen Logo-Grundfunktionen muß das Anführungszeichen jedoch ohne Zwischenraum *unmittelbar* vor dem jeweiligen Wort stehen.

4.2.2 Grundoperationen mit Wörtern

Fundamental für das Arbeiten mit Wörtern sind Grundfunktionen, mit denen man sie zerlegen kann *(Analyse)* und Funktionen, mit denen man sie zusammensetzen kann *(Synthese)*.

Die Logo-Grundfunktionen zum Zerlegen von Wörtern sind ERSTES (kurz ER), OHNEERSTES (kurz OE), LETZTES (kurz LZ) und OHNELETZTES (kurz OL). Hierzu einige Beispiele:

ERSTES "XYZ
ERGEBNIS: X

OHNEERSTES "357
ERGEBNIS: 57

LETZTES "ABC
ERGEBNIS: C

OHNELETZTES "REUTLINGEN
ERGEBNIS: REUTLINGE

(Wie man sieht, ist OHNELETZTES eine gebräuchliche Operation in der schwäbischen Sprache).

Mit Hilfe dieser Grundoperationen können wir weitere Funktionen zur Verarbeitung von Zeichenketten schreiben. Zum Beispiel:

```
PR ZWEITES  :W
 RUECKGABE  ERSTES OHNEERSTES :W
ENDE
```

(Der formale Parameter :W ist mit einem aktuellen Wort zu bestücken).

Probieren wir es aus:

ZWEITES "XYZ
ERGEBNIS: Y

Der Auswertungsprozess von Logo verläuft dabei folgendermaßen:

ZWEITES "XYZ = ERSTES OHNEERSTES "XYZ
= ERSTES "YZ
= Y

ZWEITES "987654321
ERGEBNIS: 8

ZWEITES "X
ERSTES MAG NICHT ALS EINGABE

Was ist passiert? Lassen Sie uns genau nachschauen, wie ZWEITES die Eingabe verarbeitet hat. Zunächst wird OHNEERSTES "X ausgewertet. Als Ergebnis erhalten wir das **leere Wort**, das aus gar keinem Zeichen besteht. Damit kann die Funktion ERSTES, die jetzt mit der weiteren Auswertung an der Reihe ist, aber nichts anfangen. Deshalb die Fehlermeldung. Bei dieser Fehlermeldung wird hinter dem

Wort MAG der ungeliebte Eingabewert ausgedruckt. Dieser ist im obigen Fall das leere Wort. Daher die Lücke hinter MAG.

Es ist im allgemeinen vorzuziehen, daß derartige Fehler in der Prozedur selber abgefangen werden. Hier ein Beispiel für eine verbesserte Version:

```
PR ZWEITES :W
 WENN  (  :W =  "  )  FEHLER.BEI.ZWEITES AUSSTIEG
 WENN (OHNEERSTES  :W  =  " )   FEHLER.BEI.ZWEITES AUSSTIEG
 RUECKGABE  ERSTES  OHNEERSTES  :W
ENDE

PR FEHLER.BEI.ZWEITES
 DRUCKE  [ DAS BEIM AUFRUF VON ZWEITES EINGEGEBENE WORT IST ZU KURZ ]
ENDE
```

Hierzu noch einige Kommentare:

- Die zweite Zeile von ZWEITES (WENN :W = " ...) ist ein Beispiel dafür, daß man das DANN hinter einem WENN auch weglassen kann.
- Falls das eingegebene Wort das leere Wort war, so wird die Prozedur FEHLER.BEI.ZWEITES aufgerufen, die nur die entsprechende Fehlermeldung ausdruckt und dann wieder zu ZWEITES zurückkehrt. Der als nächstes auszuführende Logo-Grundbefehl AUSSTIEG bewirkt, daß Logo die gesamte Arbeit abbricht und in die Ebene des Direktausführungsmodus zurückkehrt.
- Die nächste Zeile der Funktion ZWEITES wird also nur erreicht, wenn :W nicht das leere Wort war. Es hätte aber immer noch sein können, daß :W aus nur einem Zeichen besteht, das heißt, daß OHNEERSTES :W leer ist. Auch in diesem Fall wird eine entsprechende Fehlermeldung ausgedruckt und die Bearbeitung von ZWEITES abgebrochen.
- Wenn aber die beiden ersten Tests bestanden wurden, kann schließlich die eigentliche Arbeit, nämlich die Rückgabe des zweiten Zeichens von :W, stattfinden.

Manchmal möchte man vermeiden, daß mit dem Auftreten einer **Fehlerbedingung** die gesamte Arbeit abgebrochen wird, wie es im obigen Beispiel durch den Befehl AUSSTIEG geschah. Ein Möglichkeit, den Abbruch zu vermeiden, könnte zum Beispiel darin bestehen, daß man den Befehl AUSSTIEG durch die Rückgabe des leeren Wortes oder eines speziellen Sonderzeichens ersetzt, das sonst nicht vorkommt. Solche Lösungen bergen aber ein erhöhtes Fehlerrisiko in sich. Der Programmierer muß dann in allen Prozeduren, die auf ZWEITES zurückgreifen, prüfen, ob ein solches Sonderzeichen zurückgegeben wurde.

Aufgabe 4.1: Schreiben Sie je eine Funktion DRITTES und VIERTES.

Aufgabe 4.2: Schreiben Sie je eine Funktion VORLETZTES und VORVORLETZTES.

Der Synthese von Zeichenketten dient die Logo-Grundfunktion WORT. Zunächst einige Beispiele:

WORT "ABC "DE
ERGEBNIS: ABCDE

WORT "XQR "7
ERGEBNIS: XQR7

WORT ERSTES "OFFENBURG LETZTES "OFFENBURG
ERGEBNIS: OG

WORT "T. (WORT "L ".C) (4.1)
ERGEBNIS: T.L.C

WORT setzt also die beiden eingegebenen Zeichenketten zu einer einzigen Zeichenkette zusammen.

Als kleine Anwendung wollen wir uns einmal ansehen, wie man die Logo-Grundoperationen für ein Konjugationsprogramm einsetzen kann. Allerdings soll hier nur der einfachste Fall, nämlich nur die **regelmäßige Konjugation** im Präsens Indikativ, betrachtet werden.

Wir wollen die Konjugation am Beispiel des Wortes GEHEN entwickeln. GEHEN ist die Infinitiv-Form. Zum Zwecke der Konjugation werden zunächst die beiden letzten Zeichen entfernt; man erhält so den sogenannten Stamm des Verbs. An diesen Stamm wird dann, dem jeweiligen Personalpronomen (ICH, DU, ER / SIE / ES, WIR, IHR, SIE) entsprechend, eine Endung hinzugefügt. Ein Problem ergibt sich aus der Doppeldeutigkeit des Wortes SIE. Wie wollen deshalb im folgenden der Einfachheit halber entweder SIE.SINGULAR oder SIE.PLURAL an Stelle von SIE schreiben.

Dieser verbalen Beschreibung entnehmen wir schon die Hilfsfunktionen, die wir zur Konjugation benötigen:

```
PR STAMM  :VERB.IM.INFINITIV
 RUECKGABE  OHNELETZTES  OHNELETZTES  :VERB.IM.INFINITIV
ENDE

PR ENDUNG  :PERSONALPRONOMEN
 WENN  :PERSONALPRONOMEN  =  "ICH RUECKGABE  "E
 WENN  :PERSONALPRONOMEN  =  "DU  RUECKGABE  "ST
 WENN  :PERSONALPRONOMEN  =  "ER  RUECKGABE  "T
 WENN  :PERSONALPRONOMEN  =  "SIE.SINGULAR  RUECKGABE  "T
 WENN  :PERSONALPRONOMEN  =  "ES  RUECKGABE  "T
 WENN  :PERSONALPRONOMEN  =  "WIR RUECKGABE  "EN
 WENN  :PERSONALPRONOMEN  =  "IHR RUECKGABE  "T
 WENN  :PERSONALPRONOMEN  =  "SIE.PLURAL RUECKGABE "EN
ENDE
```

Mit diesen Hilfsprozeduren lautet nun unsere Konjugationsfunktion:

```
PR REGELMAESSIGE.KONJUGATION  :PERSONALPRONOMEN  :VERB.IM.INFINITIV
 RUECKGABE WORT (STAMM  :VERB.IM.INFINITIV ) (ENDUNG  :PERSONALPRONOMEN )
ENDE
```

Beispiele:

REGELMAESSIGE.KONJUGATION "ER "SCHREIBEN
ERGEBNIS: SCHREIBT

REGELMAESSIGE.KONJUGATION "SIE.SINGULAR "SCHWIMMEN
ERGEBNIS: SCHWIMMT

REGELMAESSIGE.KONJUGATION "DU "HELFEN
ERGEBNIS: HELFST

In diesem Stadium haben wir noch kein Instrumentarium, um festzustellen, ob ein bestimmtes Verb regelmäßig zu konjugieren ist oder nicht.

Wie Sie sehen, wurden oben ziemlich lange Namen für die Prozedur und die Parameter verwendet. Der Nachteil davon ist, daß man viel schreiben muß; der Vorteil besteht darin, daß sich die Prozedur praktisch selber dokumentiert. Es besteht keine Notwendigkeit für Kommentare, die natürlich in Logo auch möglich sind. Das Startzeichen für einen Kommentar ist das Semikolon. Der gesamte Text auf der (logischen) Zeile hinter dem **Semikolon** wird als **Kommentar** interpretiert. (Beim LCSI Logo muß der Kommentar hinter dem Semikolon in eckigen Klammern stehen).

Kommentare haben unter Umständen den Nachteil, schnell zu veralten und bei Änderungen von Programmen oder Prozeduren vergessen zu werden. Natürlich hätte man die obige Konjugationsprozedur auch folgendermaßen schreiben können:

```
PR R.K  :PP :V
 ;  R.K = REGELMAESSIGE KONJUGATION
 ;  :PP = PERSONALPRONOMEN
 ;  :V  = VERB (MUSS IM INFINITIV EINGEGEBEN WERDEN!)
 RG  WORT ( STAMM  :V )  ( ENDUNG  :PP )
ENDE
```

Die ersten drei Zeilen der Prozedur R.K sind also Kommentare. Sie spielen für die Auswertung der Funktion keine Rolle.

Die Klammern sind übrigens sowohl in der ersten wie auch in der zweiten Version entbehrlich. Sie sind nur aus Gründen der besseren optischen Gliederung hinzugefügt worden.

Wenn einem, besonders für den interaktiven Betrieb, die Bezeichnungen der ersten Version zu lang sind, kann man sich ja zum Beispiel auch Kurzformen schreiben, etwa wie folgt:

```
PR R.K  :PP  :V
 RUECKGABE REGELMAESSIGE.KONJUGATION  :PP  :V
ENDE
```

Die Dokumentationspflicht liegt dann nach wie vor bei der ausführlichen Prozedur REGELMAESSIGE.KONJUGATION; und dort ist die gesamte Dokumentation ja auch zu finden.

Aufgabe 4.3: Schreiben Sie Funktionen zur regelmäßigen Deklination von Hauptwörtern.

Manchmal möchte man mehr als zwei Zeichenketten zusammensetzen. In (4.1) sind zum Beispiel insgesamt die drei Zeichenketten T. L und .C zusammengesetzt worden. Man kann dies natürlich immer durch sukzessives Zusammensetzen von jeweils zwei Wörtern erreichen, wie in (4.1) geschehen. Logo verfügt aber, wie wir im nächsten Abschnitt sehen werden, über eine sehr angenehme Eigenschaft, die diese Aufgabe erleichtert.

4.2.3 Logo-Prozeduren, die eine variable Anzahl von Parametern zulassen

Versuchen Sie folgendes:

```
( WORT "DONAU "DAMPF "SCHIFFAHRTS "KAPITAEN  )
ERGEBNIS: DONAUDAMPFSCHIFFAHRTSKAPITAEN
```

Dies ist ein Beispiel dafür, daß WORT auch mehr als zwei Eingabewerte verarbeiten kann. Damit Logo jedoch weiß, wann die Eingabe abgeschlossen ist, muß links von WORT und rechts vom letzten Eingabewert jeweils eine runde Klammer stehen.

Natürlich kann man die runden Klammern auch verwenden, wenn man die Funktion WORT mit nur zwei Eingabewerten verwendet. Dies geschieht zum Beispiel gelegentlich aus Gründen der besseren optischen Gliederung.

```
(WORT "NUR "ZWEI  )
ERGEBNIS: NURZWEI
```

Man kann WORT, wenn man will, sogar mit nur einem Parameter aufrufen; dann benötigt man allerdings wieder die runden Klammern:

```
(WORT "WARUM  )
ERGEBNIS: WARUM
```

Merken Sie sich das folgende Beispiel gut:

```
(WORT "A "B "C)
```

Anstelle des gewünschten Ergebnisses ABC kommt die Fehlermeldung

WORT BRAUCHT MEHR EINGABEN (beim deutschen MIT Logo für den Apple II)
WORT WILL MEHR DATEN (beim deutschen Logo für den Commodore 64)

Logo will damit sagen, daß mit den Klammern etwas nicht stimmt. Erinnern Sie sich: **ein Wort wird stets durch das Leerzeichen beendet**. Der letzte aktuelle Parameter in obigem Aufruf ist also nicht der Buchstabe C, sondern das aus zwei Zeichen bestehende Wort C), das heißt: der Buchstabe C gefolgt von der runden schließenden Klammer. Damit fehlt Logo die rechte Klammer, die den gesamten Aufruf abschließt. Der Aufruf hätte so geschrieben werden müssen:

```
(WORT "A "B "C   )
```

Hier muß eine Leerstelle stehen!

Die Funktion WORT läßt also eine variable Anzahl von Eingabewerten zu. Da der Fall mit zwei Eingabewerten in der Praxis wohl am weitaus häufigsten vorkommen dürfte, wurde er zum "Normalfall" deklariert, bei dem man sich die runden Klammern schenken kann. Der Voreinstellungswert (Standardwert; englisch: default value) für die Parameterzahl von WORT ist also 2.

Neben WORT gibt es in Logo eine Reihe weiterer Grundwörter, die eine variable Anzahl von Parametern zulassen. Auch sie haben jeweils einen Voreinstellungswert für die "normale" Anzahl von Parametern. Will man von diesem Standardwert abweichen, so muß man, wie bei WORT, runde Klammern benutzen. Die folgende Tabelle gibt einen Überblick über diese Grundbefehle, auf die zum Teil später noch genauer eingegangen wird.

Logo-Grundwort	Standardwerte für die Anzahl der Parameter
ALLE?	2
DRUCKE	1
DRUCKEZEILE	1
EINES?	2
LISTE	2
SATZ	2
WORT	2

Ein Beispiel:

(DRUCKE "HEUTE "IST "EIN "SCHOENER "TAG)
HEUTEISTEINSCHOENERTAG

4.2.4 Die Einbettung von Leerzeichen in Wörter

Dies ist ein Sonderfall beim Arbeiten mit Wörtern, der nur vorsichtig und auch nur im Zusammenhang mit den Druckbefehlen verwendet werden sollte.

Im letzten Beispiel stört, daß die Wörter alle zusammengeschoben wurden. Man hätte natürlich den Ausdruck auch folgendermaßen veranlassen können:

DRUCKE [HEUTE IST EIN SCHOENER TAG]
HEUTE IST EIN SCHOENER TAG

Manchmal möchte man das Auszudruckende aber doch in der Form einer einzigen Zeichenkette schreiben. Man kann Leerzeichen in ein Wort einbetten, indem man das Wort links und rechts mit einfachen Anführungszeichen versieht. Zum Beispiel:

DRUCKE "'HEUTE IST EIN SCHOENER TAG'
HEUTE IST EIN SCHOENER TAG

Das aus einem einzigen Leerzeichen bestehende Wort sieht zum Beispiel so aus: ' '. Wenn es nicht als Prozedurname interpretiert werden soll, muß ihm ein (doppeltes) Anführungszeichen vorangestellt werden. Ein weiteres Beispiel:

(DRUCKE 3 "' ' QW 3)
3 1.73205

Wörter mit eingebetteten Leerzeichen sind letztlich aber doch Fremdkörper in Logo. Denn die Hauptfunktion des Leerzeichens besteht ja gerade darin, Wörter zu trennen. Erfahrungsgemäß treten früher oder später Probleme mit diesen beiden unverträglichen Verwendungsformen des Leerzeichens auf. Man sollte Wörter mit eingebetteten Leerzeichen deshalb nur sparsam verwenden und vorsichtig mit ihnen umgehen. Im obigen Beispiel hätte man zum Beispiel lieber mit den Hilfsprozeduren der Drucke-Stellengerecht-Umgebung (siehe Abschnitt 9.3) arbeiten oder die folgende Funktion verwenden sollen.

```
PR  LEERWORT
 RUECKGABE  ( WORT  ZEICHEN 32  ZEICHEN 32  ZEICHEN 32 )
ENDE
```

ZEICHEN 32 ist das Zeichen mit ASCII-Wert 32, also das Leerzeichen. Die deutsche Logo-Version für den Apple II kennt auch das Grundwort CHAR (= ZEICHEN). Eine kleine Verallgemeinerung der Funktion LEERWORT ist in Kapitel 9, Abschnitt 9.3 gegeben.

Der entsprechende Aufruf würde nun lauten:

(DRUCKE 3 LEERWORT QW 3)

Besonders im Zusammenhang mit Problemen der fortgeschrittenen Logo-Programmierung machen Wörter, die Leerzeichen enthalten, immer wieder Ärger (siehe zum Beispiel: die Pretty-Printing-Umgebung in Abschnitt 9.4).

Außerdem unterscheiden sich die verschiedenen Logo-Versionen sehr stark in der Methode, Leerzeichen in Wörter einzubetten. Im LCSI Logo kann man Leerzeichen (und andere Zeichen, die normalerweise als Trennzeichen interpretiert werden) dadurch in Wörter einbetten, daß man ihnen das backslash-Zeichen (\ ) voranstellt. Ein Beispiel (in LCSI Logo):

PRINT "GUTEN\ TAG
GUTEN TAG

4.2.5 Wörter als Namen

Die einzigen **Variablen**, mit denen wir bisher gearbeitet haben, waren die formalen Eingabeparameter von Funktionen und Prozeduren. Wir konnten sie durch aktuelle Parameter ersetzen, an aufrufende Prozeduren übergeben, in arithmetischen Termen verarbeiten und schließlich als Funktionswerte zurückgeben.

Wir haben aber bisher (mit Ausnahme der Ersetzung von formalen Parametern durch aktuelle Parameter) noch keine Methode kennengelernt, um einer Variablen explizit einen bestimmten Wert zuzuweisen.

Hierbei spielen Wörter eine entscheidende Rolle. Jedes Wort kann als (Variablen-) Name gedeutet werden.

Um nicht von Nebeneffekten gestört zu werden, geben wir vor der Diskussion der nächsten Beispiele ADE ein. Das heißt, wir beginnen auf einer "tabula rasa". Durch den Befehl

SETZE "X 3.14

wird das Wort X als **Name** einer Variablen definiert und dieser Variablen der Wert 3.14 zugewiesen. Den **Wert** einer Variablen bekommt man mit Hilfe der Logo-Grundfunktion WERT. Zum Beispiel:

WERT "X
ERGEBNIS: 3.14

Jetzt können wir auch das Geheimnis des Doppelpunktes (englisch: colon) lüften: :X ist nur eine Abkürzung für WERT "X . Probieren Sie es aus:

:X
ERGEBNIS: 3.14

:X - 1
ERGEBNIS: 2.14

(WERT "X) * :X
ERGEBNIS: 9.8596

Der Wert einer Variablen muß nicht etwa eine Zahl sein. Er kann einen beliebigen, in Logo definierbaren Datentyp besitzen. Beispiel:

SETZE "A "FRITZ
WERT "A
ERGEBNIS: FRITZ

Im Gegensatz zu fast allen anderen Programmiersprachen wird in Lisp/Logo zwischen dem Namen und dem Wert einer Variablen unterschieden. Will man zum Beispiel den Wert der Variablen X um 1 erhöhen, so schreibt man in

BASIC:	LET X = X+1	oder auch nur: X = X+1
Pascal:	X := X+1	
Logo:	SETZE "X WERT "X + 1	beziehungsweise
	SETZE "X :X+1	

In BASIC, Pascal und den meisten anderen Sprachen ist auf der rechten Seite des Gleichheitszeichens mit X der Wert der Variablen X gemeint, während auf der linken Seite der Name dieser Variablen gemeint ist.

Man kann die oben in verschiedenen Sprachversionen dargestellte Operation so lesen: Weise der Variablen, die den Namen X hat, den um eins erhöhten Wert der Variablen des Namens X zu.

Die Tatsache, daß Logo zwischen dem Namen und dem Wert von Variablen unterscheidet, kann kaum hoch genug eingeschätzt werden. Dadurch werden Vorgehensweisen möglich, die in anderen Sprachen schlicht unmöglich sind. Insbesondere gehört dazu:

- die **Iteration der Wertfunktion** (damit ist die Tatsache gemeint, daß der Wert einer Variablen wieder ein Variablenname sein kann, u.s.w.);

- die Möglichkeit, im Dialog mit dem Computer (interaktiv) Variable zu erzeugen und mit diesen Variablen weiterzuarbeiten.

Dazu einige Beispiele:

```
SETZE "A "B
WERT "A
ERGEBNIS: B
```

```
SETZE "B "C
WERT "B
ERGEBNIS: C
```

```
WERT WERT "A
ERGEBNIS: C
```

```
SETZE WERT "B "D
WERT "C
ERGEBNIS: D
```

(Hier wurde die Variable, die als Wert von B vorkommt, also C, mit dem Wort D belegt).

```
SETZE WERT "C WERT "A
WERT "D
ERGEBNIS: B
```

(WERT "C ist D, und diese Variable wurde mit dem Wert der Variablen A, also mit B belegt).

```
SETZE "E "F
SETZE WERT "E WERT "B
```

Wir können uns das Beziehungsgeflecht zwischen den verschiedenen Variablen auch folgendermaßen graphisch veranschaulichen:

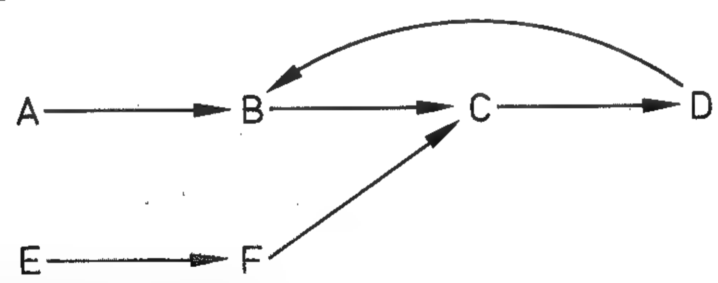


Abbildung 4.1: zum Variablenkonzept von Logo

Ein (Variablen-) Name ist also nichts anderes als ein Punkt in einem Netz, die WERT-Funktion entspricht einem Pfeil, der auf ein bestimmtes Logo-Objekt zeigt, und der Wert einer Variablen ist das Objekt, auf das der Pfeil zeigt.

Es ist also völlig natürlich, die Wertfunktion zu iterieren:

```
WERT WERT WERT "A
ERGEBNIS: D
```

(Die Wertfunktion läßt sich allerdings nur in ihrer ausführlichen Form iterieren. Man kann zum Beispiel nicht :::A an Stelle von WERT WERT WERT "A schreiben).

Schließlich sei noch ansatzweise als Anwendungsmöglichkeit des Variablenkonzepts von Logo das **Wörterbuch-Problem** beschrieben:

Stellen Sie sich vor, Sie wollen ein deutsch/englisches Wörterbuch aufbauen, das es Ihnen dann erlaubt, bei Eingabe eines deutschen Suchbegriffs durch eine entsprechende Suchfunktion das entsprechende englische Wort zurückzugeben.

Eine typische Problemlösung in einer konventionellen Sprache könnte zum Beispiel so aussehen, daß man eine zweidimensionale Tabelle aufbaut, wo jeweils in der ersten Spalte das deutsche und in der zweiten Spalte das zugehörige englische Wort steht; etwa so:

```
APFEL    APPLE
AUTO     CAR
BAUM     TREE
HAUS     HOUSE
STUHL    CHAIR
TISCH    TABLE
...      ...
```

Wenn nun die Tabelle erstellt ist, muß man eine Suchprozedur schreiben, die im ersten Ansatz etwa folgendermaßen lauten könnte:

```
SUCHPROZEDUR
 BEGINN
  GIB EINEN SUCHBEGRIFF EIN.
  FUER I=1 BIS ZUM ENDE DER TABELLE TUE FOLGENDES:
    WENN SUCHBEGRIFF = I-TER TABELLENWERT (1. SPALTE),
         DANN GIB DEN I-TEN TABELLENWERT (2. SPALTE) ZURUECK.
ENDE
```

Dies ist eine Formulierung in einer fiktiven Sprache, die sich aber leicht in Pascal umsetzen ließe.

Auch in Logo könnten wir das Problem auf diese Weise lösen. Das Variablenkonzept von Logo eröffnet aber noch andere Möglichkeiten; so zum Beispiel die folgende elegante und außerordentlich laufzeiteffiziente Lösung.

Wir fassen das deutsche Wort als Namen einer Variablen auf, deren Wert das zugehörige englische Wort ist:

```
SETZE  "APFEL  "APPLE
SETZE  "AUTO   "CAR
SETZE  "BAUM   "TREE
SETZE  "HAUS   "HOUSE
SETZE  "STUHL  "CHAIR
SETZE  "TISCH  "TABLE
   ...
```

Wenn wir jetzt ein Wort, zum Beispiel STUHL, übersetzt haben wollen, brauchen wir uns nur den Wert von STUHL geben zu lassen.

WERT "STUHL (beziehungsweise :STUHL)
ERGEBNIS: CHAIR

WERT "BAUM
ERGEBNIS: TREE

Was den deutschen Wörtern recht ist, ist den englischen billig:

SETZE "APPLE "APFEL
SETZE "CAR "AUTO
...
WERT "CAR
ERGEBNIS: AUTO

Wir können also beliebig hin- und herübersetzen. Auch folgendes ist möglich:

WERT WERT "TISCH
ERGEBNIS: TISCH

Man kann das Logo-Grundwort SETZE auch innerhalb von Prozeduren verwenden. So hätten wir zum Beispiel im obigen Falle auch Zuordnungsprozeduren schreiben können:

```
PR DEUTSCH.ENGLISCH
 SETZE "APFEL  "APPLE
 SETZE "AUTO   "CAR
 SETZE "BAUM   "TREE
 SETZE "HAUS   "HOUSE
 SETZE "STUHL  "CHAIR
 SETZE "TISCH  "TABLE
  ...
ENDE
```

```
PR ENGLISCH.DEUTSCH
 SETZE "APPLE  "APFEL
 SETZE "CAR    "AUTO
 SETZE "TREE   "BAUM
  ...
ENDE
```

Der Übersetzungsvorgang könnte nun in die folgende kleine Prozedur eingebettet werden:

```
PR UEBERSETZUNGS.DIALOG
 DRUCKE [ GEBEN SIE DAS ZU UEBERSETZENDE WORT EIN: ]
 SETZE "SUCHBEGRIFF ERSTES EINGABE
 ( DRUCKEZEILE "UEBERSETZUNG: WERT "SUCHBEGRIFF )
ENDE
```

Das darin enthaltene Logo-Grundwort EINGABE wird in Abschnitt 4.5 im Detail beschrieben. An dieser Stelle brauchen wir uns nur zu merken, daß ERSTES EINGABE das eingegebene Wort ist, mit dem die Variable SUCHBEGRIFF belegt wird.

Weitere Anwendungen der Übersetzungs-Problematik werden in Kapitel 8 behandelt.

4.3 Listen

Die Arbeit mit Listen ist das herausragendste Merkmal von Logo. **Lisp**, die Sprache, aus der heraus Logo sich entwickelt hat, steht für List Processing Language, also listenverarbeitende Sprache.

In Logo bzw. Lisp ist eine Liste ein Objekt, das aus einer beliebigen (endlichen) Anzahl von Elementen besteht; diese Elemente können Zahlen, Wörter oder selbst wieder Listen sein. Syntaktisch werden die Elemente einer Liste in Logo durch eckige Klammern zusammengefaßt; die einzelnen Listenelemente werden, durch Leerzeichen getrennt, aneinandergereiht.

Beispiele für Listen:

[X 128 A3 XYZ] ist eine Liste aus vier Elementen, dem Wort X, der Zahl 128, dem Wort A3 und dem Wort XYZ.

Die Liste [4 [A B [X] D] F7] besteht aus drei Elementen: der Zahl 4, der Teilliste [A B [X] D] und dem Wort F7.

[ ] ist die **leere Liste**, die kein Element enthält.

Die Liste ist ein strukturierter, dynamischer, rekursiver Datentyp. Was bedeuten diese Stichworte im einzelnen?

- **Strukturierbarkeit**: der Inhalt einer Liste läßt sich beliebig gliedern, hierarchisieren und auf beliebige Weise anordnen.
- **Dynamik**: Inhalt und Umfang von Listen können im Laufe ihrer Bearbeitung laufend wachsen und schrumpfen. Besonders die Möglichkeit des Schrumpfens gewährleistet, daß man bei der Bearbeitung von Listen keinen unnötigen Ballast mit sich herumzuschleppen braucht.
- **Rekursivität**: der Datentyp der Liste "enthält sich selbst"; Listen können wieder Listen als Elemente enthalten.

4.3.1 Grundoperationen der Listenverarbeitung

Einige der listenverarbeitenden Grundfunktionen von Logo haben wir in anderem Zusammenhang schon kennengelernt. Es handelt sich um die Logo-Grundwörter ERSTES (kurz ER), LETZTES (kurz LZ), OHNEERSTES (kurz OE) und OHNELETZTES (kurz OL). Ihre Wirkung auf Listen entspricht sinngemäß ihrer Wirkung auf Wörter. Einige Beispiele:

```
ERSTES [ A 1 B 2 C 3 ]
ERGEBNIS: A

OHNEERSTES [ A 1 B 2 C 3 ]
ERGEBNIS: [ 1 B 2 C 3 ]

OHNEERSTES [ A ]
ERGEBNIS: [ ]        (leere Liste)
```

LETZTES [X Y Z]
ERGEBNIS: Z

OHNELETZTES [X Y Z]
ERGEBNIS: [X Y]

Die Funktionen ERSTES und LETZTES geben also jeweils Elemente der Ausgangsliste zurück, während das Ergebnis der Funktionen OHNEERSTES und OHNELETZTES jeweils eine Teilliste der Ausgangsliste ist. Diese Funktionen erkennen jeweils, ob die Eingabewerte Listen oder Wörter sind und reagieren dann entsprechend.

Neben diesen Grundfunktionen zur Zerlegung von Listen verfügt Logo über die folgenden Möglichkeiten zum Aufbau von Listen:

Die Funktion LISTE fügt die Eingabewerte zu einer Liste zusammen. Ein Beispiel:

LISTE "X "Y
ERGEBNIS: [X Y]

Wie in Abschnitt 4.2.4 beschrieben, akzeptiert LISTE eine variable Anzahl von Eingabeparametern. Wenn man vom Standardwert 2 abweicht, muß man runde Klammern verwenden. Ein Beispiel:

(LISTE "A 1 "B 2 "C 3)
ERGEBNIS: [A 1 B 2 C 3]

Auf diese Weise kann man auch ein einzelnes Element in eine Liste verpacken:

(LISTE "Z) (Leerstelle zwischen dem Z und der Klammer!)
ERGEBNIS: [Z]

Die Funktion MITERSTEM (kurz ME) verlangt zwei Eingaben; zum Beispiel: MITERSTEM :A :B. Dabei kann :A ein beliebiges Logo-Objekt, :B muß eine Liste sein. Das Objekt :A wird als (neues) erstes Element in die Liste :B eingefügt. Beispiele:

MITERSTEM "X [1 2 3 4]
ERGEBNIS: [X 1 2 3 4]

MITERSTEM [A B C] [1 2]
ERGEBNIS: [[A B C] 1 2]

Die Funktion MITLETZTEM (kurz ML) entspricht sinngemäß der Funktion MITERSTEM.

MITLETZTEM [X5 B9] [A 0 Z 7]
ERGEBNIS: [A 0 Z 7 [X5 B9]]

Die Funktion SATZ bildet die **Vereinigungsliste** zweier Listen. Ein Beispiel:

SATZ [A B] [X Y Z]
ERGEBNIS: [A B X Y Z]

Auch für SATZ gelten die Konventionen aus Abschnitt 4.2.3 über eine variable Anzahl von Eingabeparametern.

(SATZ [A 1] [B 2 3] [C D 4])
ERGEBNIS: [A 1 B 2 3 C D 4]

Ist einer der Eingabewerte von SATZ ein Wort, so wird er wie eine einelementige Liste, die dieses Wort enthält, behandelt. Schauen Sie sich die nächsten Beispiele genau an und experimentieren Sie selber mit dem Grundwort SATZ.

SATZ "A [B C]
ERGEBNIS: [A B C]

SATZ [R S] "T
ERGEBNIS: [R S T]

(SATZ "A)
ERGEBNIS: [A]

(SATZ [A])
ERGEBNIS: [A]

SATZ [A] [B]
ERGEBNIS: [A B]

(SATZ [A] [B])
ERGEBNIS: [A B]

SATZ "A "B
ERGEBNIS: [A B]

4.3.2 Einige Besonderheiten bei der Listenverarbeitung

4.3.2.1 Die Syntaxanalyse von Listen

Wenn Logo die Elemente einer Liste "liest", gibt es sie so zurück als ob sie in gequoteter Form vorlägen. Durch ein Beispiel wird deutlicher, was hiermit gemeint ist:

SETZE "L [X Y Z]

ERSTES :L
ERGEBNIS: X

"X = ERSTES :L (oder auch: "X = ERSTES [X])
ERGEBNIS: WAHR

Man vergleiche damit:

"Y = ERSTES ["Y]
ERGEBNIS: FALSCH

Aber:

""Y = ERSTES ["Y]
ERGEBNIS: WAHR

Später, besonders in den Kapiteln 6 und 7, werden auch Logo-Grundwörter und Prozedurnamen als Listenelemente vorkommen. Dabei ist folgendes zu beachten: Wenn man zum Beispiel im Direktausführungsmodus das Kommando RICHTUNG eingibt, so führt Logo den Befehl aus und gibt die augenblickliche Peilrichtung des

Igels als Funktionswert zurück. Gibt man dagegen "RICHTUNG ein, so gibt Logo nur das Wort RICHTUNG zurück. Auch wenn RICHTUNG als Listenelement von Logo gelesen wird, wird es nur als Wort interpretiert. Logo führt in diesem Fall nicht das Kommando RICHTUNG aus. Hierzu einige Beispiele im Dialogbetrieb. Nehmen wir an, daß der Igel gerade in die Richtung 25 schaut.

```
RICHTUNG
ERGEBNIS: 25
```

```
"RICHTUNG
ERGEBNIS: RICHTUNG
```

```
ERSTES [ RICHTUNG ]
ERGEBNIS: RICHTUNG
```

4.3.2.2 Das Ausdrucken von Listen

Wie wir schon gesehen haben, läßt Logo die äußeren Listenklammern weg, wenn man eine Liste mit den Druckbefehlen DRUCKE (DR) oder DRUCKEZEILE (DZ) ausdruckt. Hierzu noch ein Beispiel:

```
DZ [ A [ B C ] ]
A [ B C ]
```

Diese Eigenschaft von Logo erleichtert die Handhabung der Druckbefehle in Dialogprogrammen. Gelegentlich möchte man aber, daß die äußeren Listenklammern gedruckt werden. Dies läßt sich beim MIT Logo für den Apple II Computer dadurch erreichen, daß man in Speicherzelle 108 des Computers eine 1 als Wechselschalter (englisch: flag) speichert. Der entsprechende Befehl lautet:

```
.LEGE 108 1
```

Ein Beispiel:

```
DZ [ A [ B C ]]
[ A [ B C ] ]
```

Der Befehl

```
.LEGE 108 0
```

macht diese Einstellung wieder rückgängig. Der .LEGE-Befehl kann natürlich auch innerhalb von Prozeduren verwendet werden.

4.3.3 Listen und Bäume

Wohl jeder hat schon einmal einen Entscheidungsbaum, einen Suchbaum, einen Strategiebaum, einen Stücklistenbaum oder einen Klassifizierungsbaum (zum Beispiel zum Zwecke biologischer Bestimmungsübungen) gesehen oder gar selber gezeichnet. Hier ein vereinfachtes Beispiel eines Stücklistenbaumes:

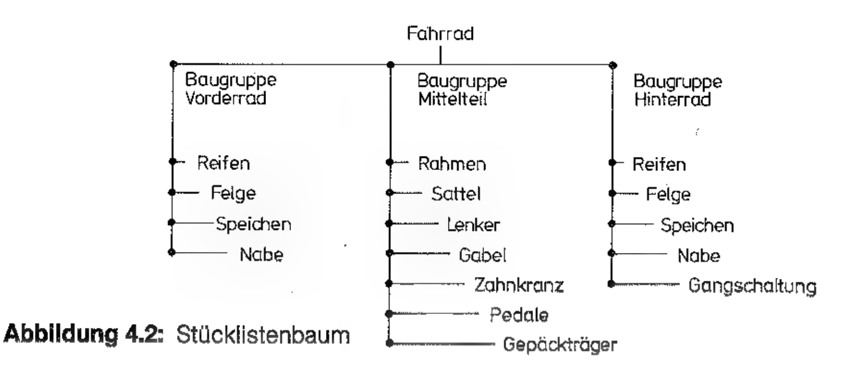


Abbildung 4.2: Stücklistenbaum

Wußten Sie, daß zur Herstellung eines Fahrrades über 1500 Einzelteile benötigt werden? Die können hier beim besten Willen nicht alle aufgeführt werden. Ich hoffe aber, Sie erkennen die Idee des Baumes. Die meisten der aufgeführten Teile sind wiederum Baugruppen, die eigentlich noch weiter aufgeschlüsselt werden müßten; so zum Beispiel die Nabe oder die Gangschaltung.

Bäume sind also nützlich, um komplexe Gliederungsstrukturen darzustellen. Sie sind ein extrem flexibles Darstellungsmittel, und letztlich ist ihre "Aussage" praktisch schlagartig von jedermann erkennbar. In den meisten Situationen, wo Bäume verwendet werden, hält man es (zu recht) für überflüssig, das Darstellungsmittel "Baum" überhaupt zu erklären.

Die enorme Flexibilität des Listenbegriffs wird nicht zuletzt daraus ersichtlich, daß sich Listen in Baumstrukturen und Bäume in Listen übersetzen lassen. Das im folgenden abgebildete Beispiel zeigt zugleich noch einmal die Wirkungsweise der Funktionen ERSTES und OHNEERSTES.

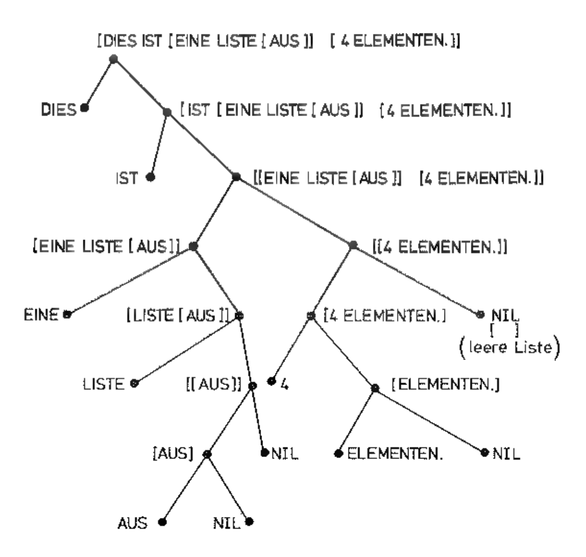
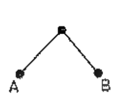


Abbildung 4.3: Umwandlung einer Liste in einen (binären) Baum

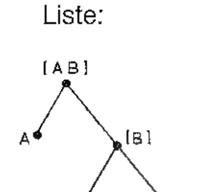
Umgekehrt lassen sich auch Bäume in Listen verwandeln:

1.Beispiel:

binärer Baum:

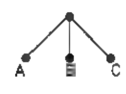


Liste:



2. Beispiel:

nicht-binärer Baum:



Liste:

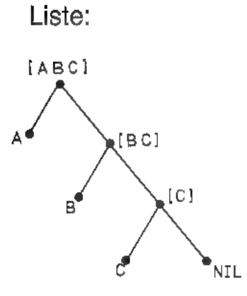


Abbildung 4.4: Umwandlung von Bäumen in Listen

Aufgabe 4.4: Stellen Sie den in Abbildung 4.2 dargestellten "Fahrrad-Baum" als Liste dar.

Aufgabe 4.5: Stellen Sie die nachfolgend gegebenen "VIERECKS.RELATIONEN" als netzartiges Beziehungsgeflecht dar.

```
PR VIERECKS.RELATIONEN
 SETZE "QUADRAT   [ RECHTWINKLIGER.DRACHEN  RAUTE  RECHTECK ]
 SETZE "RAUTE  [ DRACHEN PARALLELOGRAMM ]
 SETZE "RECHTECK  [ PARALLELOGRAMM  ACHSENSYMMETRISCHES.TRAPEZ ]
 SETZE "DRACHEN  [ VIERECK  ]
 SETZE "RECHTWINKLIGER.DRACHEN  [ DRACHEN ]
 SETZE "PARALLELOGRAMM  [ TRAPEZ ]
 SETZE "ACHSENSYMMETRISCHES.TRAPEZ  [ TRAPEZ ]
 SETZE "TRAPEZ  [ VIERECK ]
 SETZE "VIERECK  [ ]
ENDE
```

Aufgabe 4.6: Stellen Sie die nachfolgend gegebene Listenstruktur "MINI.ZOOLOGIE" als Baum dar.

```
PR MACHE.MINI.ZOOLOGIE
 SETZE  "MINI.ZOOLOGIE   [ PROTOZOA METAZOA ]
 SETZE  "PROTOZOA  [ FLAGELLATA RHIZOPODA SPOROZOA CILIATA ]
 SETZE  "RHIZOPODA   [ AMOEBE HELIOZOON RADIOLARIEN ]
 SETZE  "METAZOA   [ SPONGIA COELENTERATA PARENCHYMIA NEMATODA ________
 ______________  ARTICULATA MOLLUSCA CHORDATA ]
 SETZE  "SPONGIA   [ SCHWAMM EUSPONGIA.OFFICINALIS ]
 SETZE  "COELENTERATA   [ POLYP QUALLE KORALLE ]
 SETZE  "PARENCHYMIA   [ PLATHELMINTHES NEMERTINI ]
 SETZE  "PLATHELMINTHES   [ STRUDELWURM BANDWURM ]
 SETZE  "NEMERTINI   [ SCHNURWURM ]
 SETZE  "ARTICULATA   [ ANNELIDA ANTHROPODA ]
 SETZE  "ANNELIDA   [ REGENWURM BLUTEGEL ]
 SETZE  "ANTHROPODA   [ CRUSTACEA CHELICERATA ONYCHOPHORA TRACHEATA ]
 SETZE  "CRUSTACEA   [ KREBS ]
 SETZE  "CHELICERATA   [ SKORPION SPINNE MILBE ]
 SETZE  "TRACHEATA   [ MYRIAPODA HEXOPODA ]
 SETZE  "MYRIAPODA   [ TAUSENDFUESSLER ]
 SETZE  "HEXOPODA   [ BIENE SCHMETTERLING MUECKE WANZE LAUS ]
 SETZE  "MOLLUSCA   [ SCHNECKE MUSCHEL TINTENFISCH ]
 SETZE  "ECHINODERMATA   [ SEEIGEL SEESTERN SEEWALZE ]
 SETZE  "CHORDATA   [ ACRANIA TUNICATA VERTEBRATA ]
 SETZE  "VERTEBRATA   [ CYCLOSTAMATA PISCES AMPHIBIA REPTILIA ________
 ________________  AVES MAMMALIA ]
 SETZE  "MAMMALIA   [ MONOTREMATA MARSUPIALIA PLACENTALIA ]
 SETZE  "PISCES   [ FORELLE HAI ROCHEN ]
 SETZE  "AMPHIBIA   [ FROSCH SALAMANDER ]
 SETZE  "REPTILIA   [ KROKODIL SCHLANGE EIDECHSE ]
 SETZE  "AVES   [ ADLER MEISE SPATZ ]
 SETZE  "MONOTREMATA   [ AMEISENIGEL SCHNABELTIER ]
 SETZE  "MARSUPIALIA   [ BEUTELWOLF KAENGURUH ]
 SETZE  "PLACENTALIA   [ INSECTIVORA CHIROPTERA CARNIVORA CETACEA ______
 ________________  RODENTIA UNGULATA SIRENIA PRIMATES ]
 SETZE  "INSECTIVORA   [ IGEL SPITZMAUS MAULWURF ]
 SETZE  "CHIROPTERA   [ FLEDERMAUS ]
 SETZE  "CARNIVORA   [ BAER HUND MARDER KATZEN ]
 SETZE  "KATZEN   [ LOEWE TIGER HAUSKATZE ]
 SETZE  "CETACEA   [ ZAHNWAL BARTENWAL ]
 SETZE  "RODENTIA   [ HASE EICHHOERNCHEN BIBER MAUS ]
 SETZE  "UNGULATA   [ ELEPHANT TAPIR NASHORN PFERD SCHWEIN HIRSCH ______
 ______________  RIND ANTILOPE GIRAFFE KAMEL ]
 SETZE  "SIRENIA   [ SEEKUH ]
 SETZE  "PRIMATES   [ PROSIMIA SIMIA ]
 SETZE  "PROSIMIA   [ LEMUR ]
```

```
  SETZE "SIMIA   [ PAVIAN GIBBON ORANG-UTAN SCHIMPANSE GORILLA ________
  ___________ HOMINIDAE ]
  SETZE "HOMINIDAE   [ PITHECANTHROPUS HOMO.NEANDERTHALENSIS ________
  ______________ HOMO.SAPIENS ]
 ENDE
```

Aufgabe 4.7: (a) Stellen Sie einen Strukturbaum zum Datentyp "Adresse" auf.
(b) Setzen Sie den Baum in eine Liste um.
(c) Schreiben Sie Zugriffsfunktionen zu den einzelnen Komponenten der Adresse (zum Beispiel: NAME, NACHNAME, ORT, STRASSE, ...). Ein Eintrag in die Adressendatei könnte etwa folgendermaßen aussehen:

[007 [MAXL MEIER] [DORFSTR. 12] [SOELLACH] [BAYERN]]

```
PR NAME  :EINTRAG
 RUECKGABE  ERSTES  OHNEERSTES  :EINTRAG
ENDE

PR NACHNAME  :EINTRAG
 RUECKGABE  LETZTES  NAME  :EINTRAG
ENDE
```

(d) Erstellen Sie sich ein kleines Adressenverwaltungsprogramm.

4.4 Prädikative Funktionen ("Prüfwörter") zur Überprüfung der Korrektheit von Daten

Im Gegensatz zu vielen anderen Programmiersprachen, die eine Datenstrukturierung zulassen, ist man in Logo nicht gezwungen, den Typ eines Datenelements vor seiner Benutzung zu deklarieren. Dies hat einerseits den Vorzug einer größeren Flexibilität, birgt aber andererseits die Gefahr der illegitimen Benutzung bestimmter Datenelemente in sich. Letzteres besonders dann, wenn im Verlaufe eines Dialogs Daten vom Dialogpartner falsch eingegeben werden.

Um dieses Risiko zu vermeiden, stellt Logo eine Reihe prädikativer Funktionen, auch Prüfwörter genannt, zur Verfügung. Mit Hilfe dieser Prüfwörter lassen sich die Grund-Datentypen von Logo überprüfen. Diese prädikativen Funktionen kommen auch dann sehr gelegen, wenn man im Zweifel ist, ob eine vorgegebene syntaktische Konstruktion zu einem bestimmten Datentyp gehört oder nicht (siehe nachfolgende Beispiele).

Wie praktisch alle Logo-Grundwörter kann man auch diese Prüfwörter sowohl im Direktbetrieb als auch von Prozeduren aus benutzen.

Zur besseren Lesbarkeit sind die Prüfwörter meist durch einen besonderen Anhang gekennzeichnet. Im MIT Logo enden prädikative Funktionen mit einem Fragezeichen; im LCSI Logo mit dem Buchstaben P (für "property") am Ende des Wortes. Es gehört zum guten Programmierstil, diese Konvention auch bei selbstgeschriebenen prädikativen Funktionen einzuhalten.

Warum heißen prädikative Funktionen eigentlich so? Ein Prädikat ist eine Aussage über den Satzgegenstand. Die im folgenden zu besprechenden prädikativen Funktionen machen eine Aussage darüber, ob ein bestimmtes Objekt (der Eingabewert) von einem bestimmten Datentyp ist oder eine andere zu testende Eigenschaft hat.

Vor der Diskussion der nachfolgenden Beispiele sollte der Speicher mit ADE gelöscht werden.

ZAHL? 3
ERGEBNIS: WAHR

ZAHL? 5E14
ERGEBNIS: WAHR

ZAHL? (18 - 27) * 3.14
ERGEBNIS: WAHR

ZAHL? "XYZ
ERGEBNIS: FALSCH

WORT? "XYZ
ERGEBNIS: WAHR

WORT? "469
ERGEBNIS: WAHR

ZAHL? "469
ERGEBNIS: WAHR

ZAHL? "DREI
ERGEBNIS: FALSCH

WORT? "DREI
ERGEBNIS: WAHR

SETZE "DREI 4 (DREI ist nun eine Variable mit Wert 4).
ZAHL? WERT "DREI
ERGEBNIS: WAHR

NAME? "DREI
ERGEBNIS: WAHR

NAME? "XERXES
ERGEBNIS: FALSCH

NAME? :X gibt an, ob :X als der Name einer Variablen definiert worden ist; das heißt, ob Logo eine Variable unter dem Namen :X kennt.

WORT? [Q W E R T]
ERGEBNIS: FALSCH

LISTE? [Q W E R T]
ERGEBNIS: WAHR

LISTE? "HUGO
ERGEBNIS: FALSCH

LISTE? (LISTE "HUGO)
ERGEBNIS: WAHR
(Beachten Sie: (LISTE "HUGO) ist [HUGO]).

Wie alle Funktionen lassen sich natürlich auch die prädikativen Funktionen beliebig verknüpfen. (Die Grundwörter ALLE? und EINES? sind in Abschnitt 1.2 beschrieben).

ALLE? NAME? "X ZAHL? "UVW
ERGEBNIS: FALSCH

EINES? LISTE? [ ] ZAHL? "WSX
ERGEBNIS: WAHR

Die Existenz dieser Prüfwörter ermöglicht es uns jetzt, die Beschreibung der Grunddatentypen zugleich präzise und lapidar zu formulieren. Eine Zahl ist, was von der Funktion ZAHL? als Zahl akzeptiert wird; ein Wort ist, was von WORT? akzeptiert wird; und schließlich ist :L eine Liste, wenn LISTE? :L das Ergebnis WAHR hat.

Beispiele:

WORT? "#
ERGEBNIS: WAHR

WORT? ")
ERGEBNIS: WAHR

LISTE? (M N B)
ERGEBNIS: FALSCH

4.5 Die Erzeugung von Listen im Dialogbetrieb durch den EINGABE-Befehl

Bei konventionellen Programmiersprachen beginnt die Beschreibung der Sprache häufig mit Befehlen wie INPUT (zum Beispiel in BASIC), da dies dort die einzige Möglichkeit ist, Programme im Dialogbetrieb mit aktuellen Daten zu "bestücken". In Logo haben wir dagegen die Möglichkeit, Daten in der Form von aktuellen Parametern an Prozeduren oder Funktionen zu übergeben. Diese Form der Dateneingabe, die wir bisher fast ausschließlich praktiziert haben, ist mit der funktionalen Struktur von Logo besonders gut verträglich.

Natürlich gibt es auch in Logo die Möglichkeit, Daten in expliziter Form, etwa in einem Dialog, zu erzeugen. Man verwendet dazu den EINGABE-Befehl (abgekürzt: EG). Im englischsprachigen MIT Logo heißt der entsprechende Befehl REQUEST, im deutschen Logo für den Commodore Computer LIESLISTE (kurz: LL), im LCSI Logo READLIST (kurz RL). Im folgenden wird nur EINGABE verwendet.

Wenn der Logo-Interpreter auf das Grundwort EINGABE trifft, erwartet er eine Dateneingabe von der Tastatur. Die eingegebenen Zeichen können von (fast) beliebiger Natur sein und auch Leerzeichen enthalten. Ihre Eingabe wird mit der

RETURN-Taste beendet. EINGABE faßt alles, was eingegeben wurde, in einer Gesamtliste zusammen. Hierzu im folgenden einige Beispiele (der Deutlichkeit halber sind im folgenden Dialog die "Antworten" des Computers ausnahmsweise kursiv wiedergegeben):

EINGABE (mit <RETURN> abschließen)
1.5 3.14 2.71 17 15.8 <RETURN> (von der Tastatur einzugeben)
ERGEBNIS: [1.5 3.14 2.71 17 15.8]
(Man kann also zum Beispiel eine Reihe von Meßwerten eingeben).

Folgendes ist im Direktausführungsmodus möglich:

DRUCKEZEILE [WIE HEISST DU?]
WIE HEISST DU?
SETZE "NAME EINGABE
RUMPELSTILZCHEN (<RETURN> nicht vergessen)
WERT "NAME
ERGEBNIS: [RUMPELSTILZCHEN]

Es ist allerdings sinnvoller, diese Zeilen in leicht modifizierter Form in eine Prozedur einzubinden:

```
PR VORSTELLUNG
 DRUCKE  [ WIE HEISS DU? ]
 SETZE  "NAME  ERSTES EINGABE
 ( DRUCKEZEILE  [ GUTEN TAG,  ]   WERT  "NAME )
ENDE
```

Der Befehl EINGABE liefert in dieser Prozedur eine Liste, die den gewünschten Namen enthält. Durch das Kommando SETZE "NAME ERSTES EINGABE wird erreicht, daß die Variable NAME nur den Namen (ohne die Listenklammern) als Wert zugewiesen bekommt. Dies Prozedur ist nun folgendermaßen benutzbar:

VORSTELLUNG (ist einzutippen)
WIE HEISST DU? RUMPELSTILZCHEN (RUMPELSTILZCHEN ist einzutippen)
GUTEN TAG, RUMPELSTILZCHEN (dies druckt der Computer)

Man könnte nun fortfahren:

DRUCKEZEILE [WAS HAST DU GEKAUFT?]
WAS HAST DU GEKAUFT?
SETZE "EINKAUF EINGABE
[AEPFEL 2 KG 2.80 DM] [BIRNEN 2.5 KG 3.25 DM] ______
______ [TOMATEN 1 KG 1.80 DM] <RETURN>
:EINKAUF
ERGEBNIS: [[AEPFEL 2 KG 2.80 DM] [BIRNEN 2.5 KG 3.25 DM] _
______ [TOMATEN 1 KG 1.80 DM]]

Aufgabe 4.8: Binden Sie den letzten Dialog ebenfalls in eine entsprechende Prozedur ein.

Die Auswertung dieser Einkaufs-Information ist nun zum Beispiel auf die folgende Weise möglich:

```
PR GESAMTPREIS  :EINKAUF
 RUECKGABE (VIERTES  ERSTES   :EINKAUF) + ________________
 _______   (VIERTES  ZWEITES  :EINKAUF) + ________________
 _______   (VIERTES  DRITTES  :EINKAUF)
ENDE

PR GESAMTGEWICHT  :EINKAUF
 RUECKGABE  (ZWEITES  ERSTES  :EINKAUF) + ________________
 _______    (ZWEITES  ZWEITES  :EINKAUF) + ________________
 _______    (ZWEITES  DRITTES  :EINKAUF) ________________
ENDE
```

ZWEITES, DRITTES und VIERTES sind natürlich benutzerdefinierte Funktionen, die vorher bereitzustellen sind. Für Wörter wurde die Funktion ZWEITES in Abschnitt 4.2.2 beschrieben.

Aufgabe 4.9: Schreiben Sie Funktionen ZWEITES, DRITTES, VIERTES für Listen.

Aufgabe 4.10: Schreiben Sie "intelligentere" Versionen von ZWEITES, DRITTES, VIERTES, in denen jeweils zunächst geprüft wird, ob die aktuelle Eingabe ein Wort oder eine Liste ist und die dann in Abhängigkeit vom Prüfergebnis jeweils den richtigen Funktionswert zurückgeben. Außerdem sollte geprüft werden, ob die Eingabe lang genug ist; gegebenenfalls sollte eine Fehlermeldung ausgedruckt werden.

Diese Funktionen sollten zum Beispiel folgendes leisten:

ZWEITES [DIES IST EIN TEST]
ERGEBNIS: IST

DRITTES "NOCHEINTEST
ERGEBNIS: C

VIERTES [KURZE LISTE]
EINGABE IST ZU KURZ FUER VIERTES ! (Fehlermeldung)

Die Prozeduren GESAMTPREIS und GESAMTGEWICHT setzen voraus, daß jeweils genau drei Produkte eingekauft wurden. Elegantere Lösungen, wo Gesamtpreis und Gesamtgewicht jeweils für Einkäufe von beliebiger Größe ermittelt werden können, sind in Logo möglich. Wir werden solche Methoden in den nachfolgenden Kapiteln kennenlernen.

4.6 Weitere Möglichkeiten zur Eingabe von Daten im Dialogbetrieb

Abgesehen von der Dateneingabe in Form von aktuellen Parametern ist EINGABE zweifellos der wichtigste Befehl zur Erfassung von Daten; im folgenden seien noch einige andere Eingabemöglichkeiten erwähnt.

Alles, was eingetippt wird, wird vom Computer in einem **Tastaturpuffer** zwischengespeichert. Das Kommando TASTE gibt das "älteste" noch nicht verarbeitete Zeichen des Tastaturpuffers zurück und nimmt es dabei aus dem Puffer. Der Tasta-

turpuffer arbeitet also nach dem Prinzip **first in - first out**. Ist der Tastaturpuffer leer, so wartet TASTE auf ein (ohne <RETURN> einzugebendes) Zeichen und gibt dieses Zeichen zurück.

TASTE? gibt FALSCH oder WAHR zurück, je nachdem ob der Tastaturpuffer leer ist oder nicht.

LOESCHETASTE löscht den Eingabepuffer. Dieser Befehl ist nützlich, wenn man verhindern will, daß durch schnelles Tippen mit dem TASTE-Befehl einzulesende Antworten "im voraus" eingegeben werden.

Nur zur Vollständigkeit: Mit KNOPF und STEUER lassen sich die "Paddles", also die Spielknöpfe ansteuern und abfragen.

4.7 Benutzerdefinierte Datentypen

Im Einkaufsbeispiel in Abschnitt 4.5 haben wir implizit ein kleines Beispiel für einen benutzerdefinierten Datentyp kennengelernt. Wir könnten ihn etwa EINZELPOSTEN nennen. Ein EINZELPOSTEN ist eine Liste, die aus den folgenden fünf Objekten besteht:

(1.) einem Wort, der Warenbezeichnung;
(2.) einer Zahl, dem Gewicht in der unter (3.) angegebenen Einheit;
(3.) einem weiteren Wort, der Gewichtseinheit;
(4.) einer Zahl, dem Preis der Ware in der unter (5.) angegebenen Währung; und
(5.) noch einem Wort, der Währungseinheit.

Zur korrekten Verarbeitung eines solchen Einzelpostens muß sichergestellt sein, daß er auch in der richtigen Form eingegeben wurde. Zu diesem Zweck könnten wir eine prädikative Funktion EINZELPOSTEN? schreiben, die prüft, ob eine vorliegende Liste tatsächlich vom richtigen EINZELPOSTEN-Typ ist. Im folgenden ist ein Beispiel für eine solche Prüffunktion gegeben. Die darin verwendete Hilfsfunktion ELEMENTEZAHL, (englisch: COUNT) ist in manchen Logo-Versionen als Grundwort vorhanden, in manchen nicht. In Abschnitt 5.6.3.2 wird gezeigt, wie man sich diese Funktion selbst schreiben kann, wenn sie nicht als Logo-Grundwort vorkommt.

```
PR EINZELPOSTEN? :L
 WENN  NICHT  LISTE?  :L   RUECKGABE  "FALSCH
 WENN  NICHT  (ELEMENTEZAHL  :L )  = 5   RUECKGABE  "FALSCH
 WENN  NICHT  WORT?  ERSTES  :L   RUECKGABE  "FALSCH
 WENN  NICHT  ZAHL?  ZWEITES  :L   RUECKGABE  "FALSCH
 WENN  NICHT  WORT?  DRITTES  :L   RUECKGABE  "FALSCH
 WENN  NICHT  ZAHL?  VIERTES  :L   RUECKGABE  "FALSCH
 WENN  NICHT  WORT?  FUENFTES  :L   RUECKGABE  "FALSCH
 RUECKGABE  "WAHR
ENDE
```

EINZELPOSTEN? [PFLAUMEN 3 KG 2.80 DM]
ERGEBNIS: WAHR

EINZELPOSTEN? "BAHNHOF
ERGEBNIS: FALSCH

EINZELPOSTEN? [KIRSCHEN KG 4.20 DM]
ERGEBNIS: FALSCH

Auch die obige Prüfung ist noch verbesserungsfähig. Die Tatsache, daß die Warenbezeichnung als Wort eingegeben wurde, besagt noch nicht allzu viel. Es könnte zum Beispiel sein, daß es die eingegebene Ware gar nicht gibt. (Dies könnte etwa im Zusammenhang mit Tippfehlern passieren).

Eine verbesserte Version der Funktion EINZELPOSTEN? könnte auch solchen Aspekten noch Rechnung tragen. Man könnte etwa eine WARENLISTE anlegen und abfragen, ob der Name der eingekauften Ware in der Liste zu finden ist.

SETZE "WARENLISTE [AEPFEL BIRNEN BROT KIRSCHEN MILCH ___
__________PFLAUMEN TOMATEN]

```
PR  GUELTIGE.WARENBEZEICHNUNG?  :W
 WENN  ( ELEMENT?  :W  :WARENLISTE )   RUECKGABE "WAHR
 RUECKGABE  "FALSCH
ENDE
```

Diese Funktion kann nun an einer beliebigen Stelle in der Funktion EINZELPOSTEN? vor der vorletzten Zeile eingebaut werden:

```
PR EINZELPOSTEN?.VERBESSERT  :L
 ...       (wie in EINZELPOSTEN? )
 WENN NICHT GUELTIGE.WARENBEZEICHNUNG?  ERSTES  :L  RUECKGABE  "FALSCH
 RUECKGABE  "WAHR
ENDE
```

Ein Beispiel für die unterschiedliche Wirkung von EINZELPOSTEN? und EINZELPOSTEN?.VERBESSERT :

EINZELPOSTEN? [AUTO 920 KG 19999.00 DM]
ERGEBNIS: WAHR

EINZELPOSTEN?.VERBESSERT [AUTO 920 KG 19999.00 DM]
ERGEBNIS: FALSCH (AUTO ist nicht in der Warenliste enthalten)

Die in der Funktion GUELTIGE.WARENBEZEICHNUNG? verwendete Funktion ELEMENT? testet, ob ein bestimmtes Objekt Element der angegebenen Liste ist und gibt dementsprechend WAHR oder FALSCH zurück. Ähnlich wie ELEMENTEZAHL ist sie in den meisten Logo-Versionen als Grundwort enthalten. Ist dies nicht der Fall, kann man sich diese Funktion leicht selbst schreiben. Wir werden dies im Kapitel 5 (Abschnitt 5.6.3.2) tun. Die englisch Fassung der Funktion ELEMENT? heißt im MIT Logo MEMBER? und im LCSI Logo MEMBERP.

Der besseren Lesbarkeit halber, benutzt man gelegentlich gern die folgenden prädikativen Funktionen, mit denen überprüft werden kann, ob man es in einer speziellen Situation mit einem leeren Wort oder einer leeren Liste zu tun hat.

```
PR LEERES.WORT?  :W
 RUECKGABE  :W  =  "
ENDE

PR LEERE.LISTE?  :L
 RUECKGABE  :L  =  [ ]
ENDE
```

Diese Beispiele zeigen noch einmal (vergleiche Abschnitt 1.2), daß auch das **Gleichheitszeichen** eine Logo-Funktion ist. Der Ausdruck :W = " (und entsprechend :L = [ ]) ergibt entweder den Wert WAHR oder FALSCH, der dann von den obigen Prozeduren mit Hilfe der RUECKGABE-Befehls einfach "weitergereicht" wird.

Man kann die beiden Prozeduren auch in einer einzigen zusammenfassen:

```
PR LEER?  :X
 WENN LISTE?  :X  RUECKGABE  LEERE.LISTE?  :X
 WENN WORT?  :X   RUECKGABE  LEERES.WORT?  :X
ENDE
```

An dieser Stelle konnten naturgemäß nur allereinfachste Beispiele behandelt werden. Besonders bei komplexeren Datenstrukturen erweisen sich sowohl die in Logo "eingebauten" als auch die benutzerdefinierten prädikativen Funktionen als äußerst nützlich. Außer Lisp und Logo verfügt kaum eine Programmiersprache über prädikative Grundfunktionen zur Überprüfung von Datenstrukturen.

Kapitel 5: Die Kontrollstrukturen von Logo

Wie jede Programmiersprache verfügt auch Logo über Grundbefehle zur Bearbeitung von Daten. Diese Befehle werden normalerweise in linearer Reihenfolge, einer nach dem anderen, hintereinander ausgeführt. Daneben gibt es aber noch eine Anzahl von Grundbefehlen, die selber keine Daten verarbeiten, sondern die es ermöglichen, den Bearbeitungsablauf zu kontrollieren und insbesondere von der linearen Abarbeitung der Verarbeitungsbefehle abzuweichen.

Einige dieser Befehle haben wir schon kennengelernt. So zum Beispiel den Befehl WIEDERHOLE zur wiederholten Ausführung einer Liste von Befehlen, oder die Kontrollbefehle WENN ... DANN ... (SONST ...). Eine der wichtigsten Möglichkeiten, aus dem "linearen" Programmfluß auszubrechen, ist der Aufruf von Prozeduren und Funktionen.

In diesem Kapitel wollen wir nun einen etwas systematischeren Überblick über die Kontrollstrukturen von Logo gewinnen.

5.1 Wiederholte Ausführung von Befehlen

Der WIEDERHOLE-Befehl hat zwei Parameter: eine natürliche Zahl :N und eine Liste :L. Diese Parameter können direkt ("wörtlich") oder als Variablenwerte gegeben sein. Ein Beispiel: die Wiederholungs-Aufrufe

```
WIEDERHOLE 5 [ DRUCKEZEILE "REUTLINGEN ]
```

oder:

```
SETZE "N 5
SETZE "L [ DRUCKEZEILE "REUTLINGEN ]
WIEDERHOLE :N :L
```

bewirken beide, daß fünfmal hintereinander das Wort REUTLINGEN gedruckt wird.

Wird an Stelle der ganzen Zahl :N eine Dezimalzahl eingegeben, so wird die Wiederholungssequenz (INT :N) - mal hintereinander ausgeführt. (INT schneidet die eingegebene Zahl nach dem Dezimalpunkt ab, ohne zu runden).

5.2 Bedingte Ausführung von Befehlen

Viele Situationen verlangen von Natur aus eine oft mehrfach verschachtelte Anwendung von Kontrollwörtern der WENN ... DANN ... (SONST ...) - Gruppe.

Wir wollen dies einmal am Beispiel der Lösung **quadratischer Gleichungen** etwas eingehender studieren. Zunächst einige Vorbemerkungen. Die Gleichung

$$a * x^2 + b * x + c = 0 \tag{5.1}$$

hat in Abhängigkeit von den Koeffizienten a, b und c zwei reelle, zwei komplexe, eine doppelte, eine einfache, gar keine oder unendlich viele Lösungen. Die Abhängigkeit der Lösungsmenge von den Koeffizienten spiegelt sich im folgenden Baumdiagramm wider:

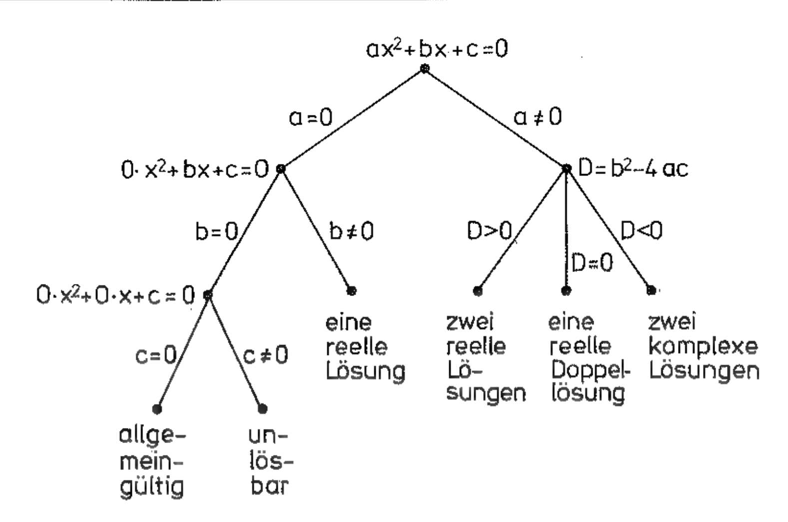


Abbildung 5.1: Fallunterscheidungen zur quadratischen Gleichung

Wenn der Koeffizient a gleich Null ist, so liegt gar keine quadratische sondern die **lineare Gleichung** $b*x + c = 0$ vor. Wir wollen diesen Gleichungstyp deswegen zuerst behandeln. Da das Problem "lineare Gleichung" von eigenständigem Interesse ist, wollen wir die Koeffizienten der Gleichung unabhängig vom Problem "quadratische Gleichung" mit a und b bezeichnen. Die lineare Gleichung lautet also

$$a*x + b = 0 \tag{5.2}$$

Als "Normalfall" wollen wir ansehen, daß die Koeffizienten a und b beide von Null verschieden sind. Die Lösung ist dann -b/a. Wenn a gleich Null und b von Null verschieden ist, so gibt es keinen Wert für x, der die Gleichung (5.2) erfüllt; die Gleichung ist dann unlösbar. Ist aber sowohl a als auch b gleich Null, so liegt die "triviale" Gleichung $0*x + 0 = 0$ vor, die für jeden Wert von x richtig ist. Gleichung (5.2) ist in diesem Fall allgemeingültig.

Bevor wir die Logo-Prozedur zur Lösung von Gleichung (5.2) schreiben, müssen wir noch eine Entscheidung darüber treffen, wie der Normalfall (Lösbarkeit) und die Fälle "unlösbar" und "allgemeingültig" behandelt werden sollen. Im Falle der Lösbarkeit ist es natürlich, die Lösung der Gleichung als Funktionswert zurückzugeben. In den Sonderfällen "unlösbar" und "allgemeingültig" könnte man zum Beispiel entsprechende "Fehler"-Meldungen drucken und die Lösungsprozedur abbrechen. Im folgenden soll jedoch so verfahren werden, daß in diesen Fällen jeweils das Wort UNLOESBAR beziehungsweise ALLGEMEINGUELTIG als Funktionswert zurückgegeben wird. Denn dies würde eine etwaige Verknüpfung unserer Lösungsfunktion mit anderen Funktionen weniger stören als der Abbruch der Funktion. In aufnehmenden Funktionen müssen diese unterschiedlichen Rückgabewerte natürlich getestet und entsprechend behandelt werden. So könnte zum Beispiel bei der Rückgabe von ALLGEMEINGUELTIG der Benutzer in einem Dialog entsprechend informiert und zur Eingabe einer beliebigen Zahl aufgefordert werden. Ihm ist

dadurch unter Umständen bei der Lösung seines Gesamtproblems, innerhalb dessen die Lösung einer linearen Gleichung vielleicht nur ein kleines Teilproblem darstellte, mehr gedient als durch den "Ausstieg" aus der Prozedur.

```
PR LINEARE.GLEICHUNG  :A  :B
 WENN  :A = 0  DANN _______________________________________
 ____ WENN  :B = 0  DANN  RUECKGABE  "ALLGEMEINGUELTIG ___________
 _____  SONST  RUECKGABE  "UNLOESBAR ______________________
 ____ SONST  RUECKGABE  ( - :B / :A )
ENDE
```

(Im Hinblick auf die Verwendung des Unterstreichungszeichens sei auf die entsprechende Konvention in Kapitel 2, Abschnitt 2.3 erinnert).

Nun zur quadratischen Gleichung (5.1). Im "Normalfall" hat diese Gleichung zwei reelle Zahlen als Lösung, nämlich

$x_1 = (-b + QW(b^2 - 4ac)) / 2a$

$x_2 = (-b - QW(b^2 - 4ac)) / 2a$

Logo-Funktionen können jeweils nur einen Funktionswert zurückgeben. Allerdings kann dieser Funktionswert auch eine Liste sein. Es ist also nur natürlich, die beiden Lösungswerte zu einer Liste zusammenzufassen und diese Liste als Funktionswert zurückgeben zu lassen. Damit die Funktionswertrückgabe in einheitlicher Weise erfolgt, sollen auch in allen anderen Fällen die Funktionswerte als Elemente einer Liste zurückgegeben werden.

```
PR QUADRATISCHE.GLEICHUNG  :A  :B  :C
 SETZE  "D  ( :B * :B - 4 * :A * :C )
 WENN  :A = 0  DANN RUECKGABE ( LISTE LINEARE.GLEICHUNG  :B  :C ) _____
 ____ SONST WENN  :D > 0  DANN  RUECKGABE  ZWEI.REELLE.LOESUNGEN_______
 _____  SONST WENN  :D = 0  DANN  RUECKGABE  DOPPELLOESUNG ___________
 ____________  SONST  RUECKGABE  KOMPLEXE.LOESUNGEN
ENDE

PR ZWEI.REELLE.LOESUNGEN
 RUECKGABE  ( LISTE ( - :B + ( QW :D ) ) / ( 2 * :A )  ___________
 _________________  ( - :B - ( QW :D ) ) / ( 2 * :A ) )
ENDE

PR DOPPELLOESUNG
 RUECKGABE ( LISTE  ( - :B ) / ( 2 * :A ) "DOPPELLOESUNG )
ENDE

PR KOMPLEXE.LOESUNGEN
 RUECKGABE ( LISTE _______________________________________________
 __________ (WORT - :B / ( 2 * :A ) "+ ( QW - :D ) / ( 2* :A ) "*I ) _
 __________ (WORT - :B / ( 2 * :A ) "- ( QW - :D ) / ( 2 * :A ) "*I ) )
ENDE
```

Eine vereinfachte Version der letzten Prozedur, bei der die komplexen Zahlen als nicht bekannt vorausgesetzt werden sollen, könnte zum Beispiel auch folgendermaßen lauten:

```
PR KOMPLEXE.LOESUNGEN.VEREINFACHT
 RUECKGABE ( LISTE  "UNLOESBAR )
ENDE
```

Einige Dialogbeispiele:

QUADRATISCHE.GLEICHUNG 1 (−5) 6
ERGEBNIS: [3.0 2.0]

QUADRATISCHE.GLEICHUNG 1 (−10) 25
ERGEBNIS: [5.0 DOPPELLOESUNG]

QUADRATISCHE.GLEICHUNG 1 0 (−2)
ERGEBNIS: [1.41421 −1.41421]

QUADRATISCHE.GLEICHUNG 1 0 1
ERGEBNIS: [0.0+1*I 0.0−1*I]

oder im Falle KOMPLEXE.LOESUNGEN.VEREINFACHT
ERGEBNIS: [UNLOESBAR]

QUADRATISCHE.GLEICHUNG 0 4 (−3)
ERGEBNIS: [0.75]

QUADRATISCHE.GLEICHUNG 0 0 7
ERGEBNIS: [UNLOESBAR]

QUADRATISCHE.GLEICHUNG 0 0 0
ERGEBNIS: [ALLGEMEINGUELTIG]

Man wird natürlich im Dialogbetrieb (außer zu Testzwecken) kaum jemals die Parameter 0 0 0 eingeben; aber QUADRATISCHE.GLEICHUNG kann auch von einer anderen Funktion aus aufgerufen werden, die zugleich die Koeffizienten (automatisch) mit aktuellen Werten bestückt. Dabei kann es durchaus auch zu einer derartigen Besetzung der Parameter kommen.

Einige ergänzende Bemerkungen zur Funktionsgruppe QUADRATISCHE.GLEICHUNG.

(1.) Die "Unterfunktionen" ZWEI.REELLE.LOESUNGEN, DOPPELLOESUNG und KOMPLEXE.LOESUNGEN sind hier ohne Eingabe-Parameter formuliert. In derartigen Fällen gilt die Regel, daß alle Variablen, die in der aufrufenden Prozedur oder Funktion (in diesem Fall QUADRATISCHE.GLEICHUNG) bekannt sind, auch in den aufgerufenen Prozeduren beziehungsweise Funktionen (hier ZWEI.REELLE.LOESUNGEN, DOPPELLOESUNG und KOMPLEXE.LOESUNGEN) bekannt sind und daß sie dieselben Werte wie in der aufrufenden Prozedur haben. Im Unterschied zur Funktion LINEARE.GLEICHUNG, die auch selbständig aufgerufen werden kann, stellen die obigen Funktionen ZWEI.REELLE.LOESUNGEN, DOPPELLOESUNG und KOMPLEXE.LOESUNGEN reine "Satelliten"-Funktionen der Funktion QUADRATISCHE.GLEICHUNG dar, deren Aufruf nur von der letzte-

ren Funktion aus sinnvoll ist. Wir werden auf diesen Sachverhalt im Kapitel "Funktionen und Prozeduren in vertiefter Behandlung" (Kapitel 6, besonders Abschnitt 6.2) intensiv zu sprechen kommen.

(2.) Der Befehl SETZE "D (:B * :B - 4 * :A * :C) in der Funktion QUADRATISCHE.GLEICHUNG stellt ein kleines Problem dar, weil die Variable D auch dann noch existiert, wenn die Funktion QUADRATISCHE.GLEICHUNG längst abgearbeitet ist. D ist, wie man auch sagt, eine globale Variable. Sie belastet den Arbeitsspeicher und könnte eventuell im Zusammenhang mit anderen Prozeduren Namenskonflikte hervorrufen. Dieses Dilemma läßt sich beheben, indem man D als lokale Variable definiert. Auch hierauf wird in Kapitel 6 über die vertiefte Behandlung von Funktionen näher eingegangen.

(3.) Der Ausdruck WENN Bedingung DANN Handlung1 (SONST Handlung2) heißt im (englischsprachigen) LCSI Logo IF Bedingung THEN Handlung1 (ELSE Handlung2). Im Unterschied zum MIT Logo müssen Handlung1 und Handlung2 als Listen eingegeben werden. Dafür können dann die Wörter THEN und gegebenenfalls ELSE entfallen. Die folgenden beiden Formulierungen sind im LCSI Logo also gleichwertig.

```
TO ABSOLUTE  :X
 IF  :X <  0 THEN [ OUTPUT  - :X ]  ELSE  [ OUTPUT  :X ]
END
```

oder:

```
TO ABSOLUTE  :X
 IF  :X < 0  [ OUTPUT  - :X  ]  [ OUTPUT  :X ]
END
```

Die Beispiele LINEARE.GLEICHUNG und QUADRATISCHE.GLEICHUNG zeigen, daß verschachtelte WENN ... DANN ... (SONST ...) - Aufrufe leicht zu unübersichtlichen Darstellungen führen können. Hierfür gibt es mehrere Abhilfemöglichkeiten:

(1.) Wenn der SONST - Zweig in der Beendigung der Prozedur resultiert, also zum Beispiel im Zusammenhang mit einer Funktionswertrückgabe, dann kann man auch nach dem DANN - Teil zu einer neuen (logischen) Zeile übergehen. Also zum Beispiel:

```
PR ABSOLUTBETRAG  :X
 WENN  :X < 0   DANN RUECKGABE   - :X  SONST  RUECKGABE  :X
ENDE
```

kann auch folgendermaßen formuliert werden:

```
PR ABSOLUTBETRAG  :X
 WENN  :X < 0  DANN RUECKGABE  - :X
 RUECKGABE  :X
ENDE
```

Ist die eingangs gemachte Einschränkung jedoch nicht erfüllt, so funktioniert diese Methode nicht; zum Beispiel:

```
PR ABSOLUT.DRUCK.1  :X
 WENN :X < 0  DANN  DRUCKEZEILE  - :X  SONST  DRUCKEZEILE  :X
ENDE

PR ABSOLUT.DRUCK.2  :X
 WENN  :X < 0  DANN  DRUCKEZEILE  - :X
 DRUCKEZEILE  :X
ENDE
```

Während ABSOLUT.DRUCK.1 stets nur den Absolutbetrag von :X ausdruckt, druckt ABSOLUT.DRUCK.2 bei negativem Wert von X zuerst den Absolutbetrag von :X und dann noch einmal :X selber aus, weil eben DRUCKEZEILE, im Gegensatz zu RUECKGABE, die Prozedur nicht beendet.

Beispiele:

ABSOLUT.DRUCK.1 (-3)
3

ABSOLUT.DRUCK.2 (-3)
3
-3

(2.) Die Verwendung der PRUEFE-Kontrollstruktur. Sie hat die folgende Syntax:

    PRUEFE Bedingung
    WENNWAHR Handlung1
    WENNFALSCH Handlung2

WENNWAHR kann als WW und WENNFALSCH als WF abgekürzt werden (englisch: TEST, IFTRUE (IFT), IFFALSE (IFF)).

Sowohl WENNWAHR ... als auch WENNFALSCH ... können dabei jeweils am Anfang einer neuen logischen Zeile stehen. WENNWAHR und WENNFALSCH reagieren jeweils auf die letzte PRUEFE-Abfrage. Eine logische Verschachtelung von PRUEFE-Sequenzen ist deshalb nur bedingt möglich.

Die Funktionen LINEARE.GLEICHUNG und QUADRATISCHE.GLEICHUNG können zum Beispiel mit Hilfe des PRUEFE-Kommandos auch folgendermaßen formuliert werden:

```
PR LINEARE.GLEICHUNG  :A  :B
 PRUEFE  :A = 0
 WENNFALSCH  RUECKGABE  ( -:B / :A )
 PRUEFE  :B = 0
 WENNWAHR  RUECKGABE  "ALLGEMEINGUELTIG
 WENNFALSCH  RUECKGABE  "UNLOESBAR
ENDE
```

```
PR QUADRATISCHE.GLEICHUNG  :A  :B  :C
 PRUEFE  :A = 0
 WENNWAHR  RUECKGABE ( LISTE LINEARE.GLEICHUNG  :B  :C )
 SETZE  "D  ( :B * :B  -  4 * :A * :C )
 PRUEFE  :D > 0
 WENNWAHR  RUECKGABE ZWEI.REELLE.LOESUNGEN
 PRUEFE  :D = 0
 WENNWAHR  RUECKGABE DOPPELLOESUNG
 RUECKGABE KOMPLEXE.LOESUNGEN
ENDE
```

Die für die ursprünglichen Versionen von LINEARE.GLEICHUNG und QUADRATISCHE.GLEICHUNG vorgesehenen "Satelliten"-Funktionen ZWEI.REELLE.LOESUNGEN, DOPPELLOESUNG und KOMPLEXE.LOESUNGEN können in den neuen Versionen unverändert verwendet werden.

5.3 Bedingungen

Unter einer Bedingung wollen wir im folgenden eine Funktion verstehen, die einen der beiden Werte WAHR oder FALSCH zurückgibt. In anderem Zusammenhang (Kapitel 4, Abschnitt 4.4) haben wir derartige Funktionen auch prädikative Funktionen (beziehungsweise Prüfwörter) genannt. Schließlich werden solche Funktionen auch häufig als **Boolesche Funktionen** bezeichnet (George Boole, englischer Mathematiker, 1815 - 1864; einer der Begründer der modernen formalen Logik). Die unterschiedlichen Bezeichnungen hängen vom Kontext ab, in dem man derartige Funktionen betrachtet.

Da wir die in Logo als Bedingung verankerten Grundwörter schon kennengelernt haben, sollen sie hier nur noch einmal systematisch, aber knapp zusammengestellt werden. Außerdem soll noch einmal bewußt gemacht werden, daß auch das Gleichheitszeichen sowie die Relationen < und > als Funktionen angesehen werden können. Denn sie liefern WAHR oder FALSCH als Funktionswerte:

1 + 2 = 3
ERGEBNIS: WAHR

4 > 6
ERGEBNIS: FALSCH

Normalerweise wird eine Funktion in Logo so dargestellt, daß man zuerst den Funktionsnamen und dann die Parameter aufschreibt. Man nennt diese Schreibweise auch die **Präfix-Notation** oder auch die Cambridge-Notation. (Cambridge, weil Lisp, das ausschließlich diese Notation benutzt, in Cambridge, Massachusetts, entwickelt wurde). In einer reinen Präfix-Sprache müßte man zum Beispiel die Summe der Zahlen 2 und 3, die man üblicherweise in der sogenannten **Infix-Notation** als 2+3 schreibt in der Form (+ 2 3) oder (ADD 2 3) schreiben. Die Gleichung 2+3=5 lautet in der Präfix-Schreibweise:

(= 5 (+ 2 3)) oder (GLEICH? 5 (ADD 2 3)).

Im Gegensatz zu Lisp verfährt Logo bei den Grundrechenarten und bei den "Funktionen" = , < , und > nach der gebräuchlicheren Infix-Notation. Dies ändert jedoch nichts an der Tatsache, daß es sich dabei um Funktionen handelt.

Weitere Bedingungen sind die bekannten Prüfwörter LISTE?, NAME?, WORT?, und ZAHL?. Dazu kommen noch die benutzerdefinierten Prüfwörter wie zum Beispiel EINZELPOSTEN? in Kapitel 4, Abschnitt 4.7.

Schließlich dienen ALLE? und EINES? der Kombination mehrerer Bedingungen. Es sei nochmals daran erinnert, daß für ALLE? und EINES? die Konvention über beliebig viele Eingabeparameter (im Zusammenhang mit der Verwendung von runden Klammern) aus Kapitel 4, Abschnitt 4.2.3 gilt.

Ein Beispiel:

(ALLE? (2 * 3 = 6) (4 < 8) (WORT? "HALLO))
ERGEBNIS: WAHR

5.4 Abbruch der Bearbeitung von Prozeduren

Der Befehl AUSSTIEG unterbricht die gesamte Aufrufkette und bewirkt die Rückkehr in die oberste Kommandoebene (Direktausführungs-Modus). Dieser Befehl ist dann nützlich, wenn ein irreparabler Fehler eingetreten ist, der die gesamte weitere Auswertung sinnlos machen würde. Ein Beispiel:

```
PR PROZENTSATZ  :GW  :PW
 WENN  :GW = 0  DANN  DRUCKE  "EINGABEFEHLER  AUSSTIEG
 RUECKGABE  ( :PW / :GW ) * 100
ENDE
```

Beispiele:

PROZENTSATZ 20 1.5
ERGEBNIS: 7.5 (in Hundertsteln; siehe allgemeine Hinweise)

PROZENTSATZ 0 30
EINGABEFEHLER

Eine weitere Art, die Bearbeitung einer Funktion zu unterbrechen, ist die Funktionswertrückgabe, also der Befehl RUECKGABE. Allerdings resultiert dies im Gegensatz zu AUSSTIEG in einem "geordneten" Verlassen der Funktion und zur Rückkehr an den Ort, von dem aus die betroffene Funktion aufgerufen wurde.

Es gibt auch viele Situationen, wo man eine Prozedur, die keine Funktion ist, auf eine derartige geordnete Weise verlassen möchte. Diesem Zweck dient das Logo-Grundwort RUECKKEHR (kurz RK). Es bewirkt, wie RUECKGABE, die Beendigung der Arbeit an der jeweiligen Prozedur und die Rückkehr zur aufrufenden Prozedur. Im Unterschied zu RUECKGABE resultiert RUECKKEHR jedoch nicht in der Übergabe eines Funktionswertes.

Ein Beispiel für die Verwendung des RUECKKEHR-Befehls ist im nächsten Abschnitt im Zusammenhang mit der ZINSESZINS-Prozedur gegeben. Weitere Beispiele folgen im Abschnitt 5.6 über Rekursion.

5.5 Der Sprungbefehl

Obwohl man es (praktisch) nie muß, kann man in Logo mit Hilfe des GEHE-Befehls (englisch: GO) zu vorgegebenen **Marken** springen. Ein Beispiel:

```
PR ZINSESZINS  :ANFANGSKAPITAL  :ZINSSATZ  :LAUFZEIT
 SETZE  "LAUFENDES.KAPITAL  :ANFANGSKAPITAL
 SETZE "LAUFINDEX  0
 SCHLEIFENBEGINN:
 WENN  ( :LAUFINDEX + 1 ) >   :LAUFZEIT  DANN RUECKKEHR
 SETZE  "LAUFENDES.KAPITAL ______________________________
 ___  ( :LAUFENDES.KAPITAL  +  :LAUFENDES.KAPITAL  * :ZINSSATZ / 100 )
 SETZE  "LAUFINDEX  ( :LAUFINDEX + 1 )
 ( DRUCKEZEILE  :LAUFINDEX  LEERWORT  :LAUFENDES.KAPITAL )
 GEHE  "SCHLEIFENBEGINN
ENDE
```

Der Markierungsname SCHLEIFENBEGINN ist völlig willkürlich. Man hätte ebenso etwa ANFANG oder XAVER oder A0 verwenden können. In der Definitionszeile muß unmittelbar anschließend an den Markierungsnamen ein Doppelpunkt (englisch: colon) folgen. Denn daran erkennt der Logo-Interpreter, daß es sich um eine Markierung und nicht etwa um einen Prozedurnamen handelt. Der Doppelpunkt ist jedoch kein Bestandteil des Markierungsnamens. (Die Hilfsfunktion LEERWORT ist in Abschnitt 4.2.4 besprochen worden).

Ein Beispiel (mit Zinssatz = 5 %, einzugeben in Hundertsteln siehe allgemeine Hinweise):

```
ZINSESZINS 200 5 10

 1   219.0
 2   220.5
 3   231.525
 4   243.101
 5   255.256
 6   268.019
 7   281.42
 8   295.491
 9   310.265
10   325.779
```

An dieser Stelle erscheint es angebracht, auf einige (längst nicht alle) Unterschiede zwischen Logo und der sonst auf Mikrocomputern oft benutzten Programmiersprache BASIC einzugehen.

In der Programmiersprache BASIC gibt es neben der (oft nicht schachtelbaren) bedingten Ausführung und dem Sprung meist keine weiteren Kontrollstrukturen. Das obige Programm könnte in BASIC etwa folgendermaßen geschrieben werden:

```
10 INPUT AK
20 INPUT ZS
30 INPUT LZ
40 LET LK=AK
50 LET LI=0
60 IF (LI+1) > LZ  THEN GOTO  110
70 LET LK = LK  +  LK * ZS / 100
80 LET LI = LI + 1
90 PRINT LI; "    "; LK
100 GOTO 60
110 END
```

Die Tatsache, daß man (abgesehen von sehr seltenen Ausnahmeversionen) in BASIC nur auf Zeilennummern springen kann, macht die Sprache sehr fehleranfällig. Denn gelegentlich ist man gezwungen, Zeilennummern oder ihre Inhalte zu ändern. Wenn man dann nicht alle Sprungadressen ändert, die auf diese Zeilennummer bezug nehmen, ist ein Fehler unvermeidbar. Besonders heimtückisch ist die Tatsache, daß derartige Fehler meist logischer und nicht syntaktischer Natur sind. Sie werden also vom BASIC-Interpreter nicht erkannt und resultieren bei etwas längeren Programmen nicht selten in einer außerordentlich aufwendigen Fehlersuche.

Man hätte auch in der obigen Logo-Prozedur ZINSESZINS mit dem EINGABE-Befehl anstelle von Eingabeparametern arbeiten können. Dann hätte die Logo-Prozedur strukturell vollständig dem BASIC-Programm entsprochen. (Natürlich noch abgesehen von der Tatsache, daß man in der Logo-Prozedur auf eine namentlich gegebene Sprungmarke springen kann und nicht auf eine Zeilennummer springen muß und daß man beliebig lange Variablennamen verwenden darf).

Entsprechendes gilt auch für sonstige BASIC-Programme: Sie lassen sich stets durch entsprechende strukturgleiche Logo-Prozeduren "simulieren". Man kann also, wenn man möchte, jedes BASIC-Programm ohne weiteres direkt in Logo nachbilden. Nur ist dies nicht empfehlenswert, da es in Logo praktisch immer bessere Lösungen gibt, die in BASIC gar nicht möglich sind.

Diese Überlegungen zeigen übrigens auch, daß das, was in BASIC als Programm bezeichnet wird, in Logo üblicherweise von Prozeduren erledigt wird. Während sich in BASIC immer nur ein einziges Programm im Arbeitsspeicher befindet, das nur mit dem stereotypen Befehl RUN ausgeführt werden kann, können in Logo viele Prozeduren im Arbeitsspeicher abgelegt und über ihren Prozedurnamen aufgerufen werden.

Abschließend noch einige allgemeine Bemerkungen zur Verwendung von Sprungbefehlen.

(1.) Programme, die viele Sprungbefehle enthalten, werden leicht außerordentlich unübersichtlich. Man sollte die Verwendung von Sprungbefehlen deshalb auf ein

Mindestmaß beschränken. Mir ist nur eine einzige außerordentlich seltene Situation bekannt, wo man auf den Sprungbefehl nicht verzichten kann. Sie wird in Kapitel 6 in den Aufgaben 6.21 bis 6.23 behandelt.

Wie jede Sprache, verfügt auch Logo über eine "Sprachstilistik". Im Sinne dieser Stilkunde wird die Verwendung von Sprungbefehlen als kein guter Programmierstil angesehen. Harold Abelson schreibt in seinem Buch "Logo for the Apple II": ... With all these possibilities, peppering Logo procedures with GOs is like pouring ketchup over caviar.

Übrigens wird in allen höheren Programmiersprachen von der Verwendung von Sprungbefehlen abgeraten. In manchen Pascal-Versionen muß man zum Beispiel erst eine "Sperrvorrichtung" in Form einer sogenannten compiler option öffnen, um überhaupt mit dem GOTO-Befehl arbeiten zu können.

Der bekannte Informatiker E.W.Dijkstra hat in der amerikanischen Fachzeitschrift "Communications of the Association for Computing Machinery", März 1968, einen vielbeachteten Beitrag unter dem Titel "GOTO Statement Considered Harmful" veröffentlicht.

(2.) Im MIT Logo darf der GEHE-Befehl nur in Prozeduren (und nicht im Direktausführungsmodus) verwendet werden. Außerdem ist es verboten, zu einer Marke zu springen, die außerhalb der Prozedur liegt, von der aus der Sprung erfolgt. Selbst wenn letzteres bei bestimmten Logo-Versionen syntaktisch erlaubt sein sollte, sollte man keinen Gebrauch davon machen, da dies den Kontrollfluß (einschließlich der Variablenbindungen) in völlig unüberschaubarer Weise durcheinander bringen würde.

5.6 Rekursion

In den bisher behandelten Beispielen ist es schon oft vorgekommen, daß eine Prozedur eine andere Prozedur aufrief. Das ist, insbesondere auch wenn es sich um Funktionen handelt, die normale Arbeitsweise in Logo. Obwohl es bisher noch nicht praktiziert wurde, ist es keineswegs verboten, daß eine Prozedur beziehungsweise eine Funktion sich selbst aufruft. Das Phänomen des Sich-selbst-Aufrufens wird als rekursives Aufrufen oder kurz als Rekursion bezeichnet.

Die Rekursion spielt in Lisp und Logo eine herausragende Rolle. Diese beiden Sprachen basieren außerordentlich stark auf rekursiven Vorgehensweisen. (Ähnliches gilt übrigens auch für die Mathematik).

5.6.1 Rekursive Prozeduren im nichtgraphischen Bereich

Wenn wir etwa die Buchstaben eines Wortes der Reihe nach ausdrucken lassen wollen, so ist es natürlich, dies mit der folgenden rekursiven Prozedur zu tun:

```
PR BUCHSTABEN  :W
 WENN  :W = "  DANN RUECKKEHR
 DRUCKEZEILE  ERSTES  :W
 BUCHSTABEN  OHNEERSTES  :W
ENDE
```

Beispiel:
BUCHSTABEN "HAUS
H
A
U
S

Schauen wir uns einmal im Detail an, wie die Prozedur BUCHSTABEN die Eingabe HAUS abarbeitet:

1. Aufruf von BUCHSTABEN mit Eingabe HAUS.
   Die Eingabe ist nicht leer.
   Also Ausdruck von H (= ERSTES :W).
   2. Aufruf von BUCHSTABEN mit Eingabe AUS (=OHNEERSTES "HAUS).
      Die Eingabe ist nicht leer.
      Also Ausdruck von A.
      3. Aufruf von BUCHSTABEN mit Eingabe US.
         Die Eingabe ist nicht leer.
         Also Ausdruck von U.
         4. Aufruf von BUCHSTABEN mit Eingabe S.
            Die Eingabe ist nicht leer.
            Also Ausdruck von S.
            5. Aufruf von BUCHSTABEN mit Eingabe (leeres Wort).
               Die Eingabe ist leer.
               Also Rückkehr zu der Stelle, von der aus dieser Aufruf erfolgte. Dies war der 4. Aufruf von BUCHSTABEN.
         Damit ist der 4. Aufruf von BUCHSTABEN völlig abgearbeitet.
         Aufgrund des üblichen Programmflusses macht Logo an der Stelle weiter, von der aus der 4. Aufruf erfolgte. Dies war der 3. Aufruf von BUCHSTABEN.
      Damit ist der 3. Aufruf von BUCHSTABEN abgearbeitet. Logo macht an der Stelle weiter, von wo aus der 3. Aufruf erfolgte. Dies war der 2. Aufruf von BUCHSTABEN.
   Damit ist der 2. Aufruf von BUCHSTABEN abgearbeitet. Logo macht an der Stelle weiter, von der aus der 2. Aufruf erfolgte. Dies war der 1. Aufruf von BUCHSTABEN.

Damit ist der 1. Aufruf von BUCHSTABEN abgearbeitet. Logo macht an der Stelle weiter von der aus der 1. Aufruf von BUCHSTABEN erfolgte.

An Stelle dieser verbalen Schilderung verwendet man zur Illustration des Programmflusses bei rekursiven Prozeduren auch Diagramme der folgenden Art. (Die Zahlen in den Kreisen stellen die Nummer des jeweiligen Aufrufs dar).

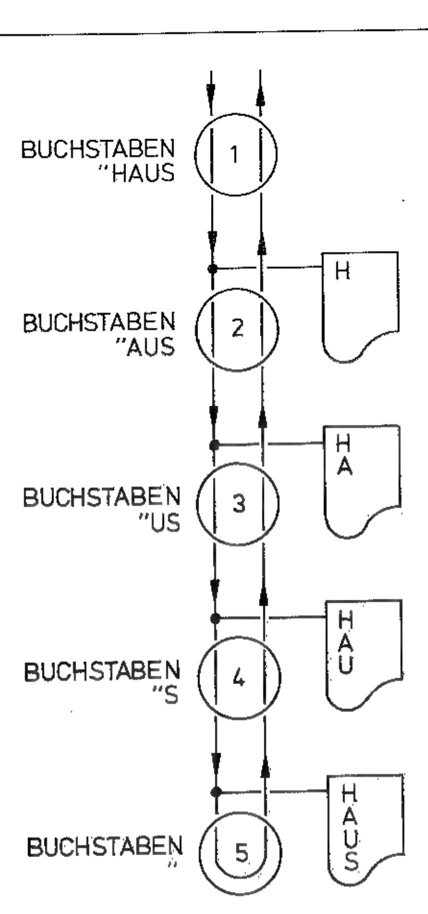


Abbildung 5.2

Nun ein weiteres Beispiel (die Zeilennummern gehören nicht zur Logo-Prozedur selbst sondern dienen nur zur Erläuterung).

```
1:  PR HOPPLA  :W
2:   WENN :W = "   DANN RUECKKEHR
3:   HOPPLA OHNEERSTES "W
4:   DRUCKEZEILE ERSTES "W
5:  ENDE
```

HOPPLA "HAUS
S
U
A
H

An Stelle von HOPPLA hätten wir also auch den Namen BUCHSTABEN.UMGE-KEHRT verwenden können.

Wir verfolgen ebenfalls den Programmfluß:

1. Aufruf von HOPPLA mit Eingabe HAUS.
   Die Eingabe ist nicht leer; also
   2. Aufruf von HOPPLA mit Eingabe AUS.
      Die Eingabe ist nicht leer; also
      3. Aufruf von HOPPLA mit Eingabe US.
         Die Eingabe ist nicht leer; also
         4. Aufruf von HOPPLA mit Eingabe S.
            Die Eingabe ist nicht leer; also
            5. Aufruf von HOPPLA mit Eingabe (leeres Wort).
               Die Eingabe ist leer; also Rückkehr zu der Stelle, von der aus der 5. Aufruf erfolgte. Dies war der 4. Aufruf, 3. Programmzeile. Sie ist nun abgearbeitet.

            Logo fährt mit der Bearbeitung des 4. Aufrufs von HOPPLA in der 4. Programmzeile fort. Die Eingabe beim 4. Aufruf war S.
            ERSTES von S ist ebenfalls S.
            Logo druckt also S aus.
            Damit ist der 4. Aufruf von HOPPLA abgearbeitet.
            Logo macht an der Stelle weiter, von der aus der 4. Aufruf erfolgte.
            Dies war der 3. Aufruf, 3. Programmzeile.
            Sie ist nun abgearbeitet.

         Logo fährt mit der Bearbeitung des 3. Aufrufs von HOPPLA in der 4. Programmzeile fort. Die Eingabe beim 3. Aufruf war US.
         Also druckt Logo U aus.
         Damit ist der 3. Aufruf von HOPPLA abgearbeitet.
         Logo macht an der Stelle weiter, von wo aus der 3. Aufruf erfolgte. Dies war der 2. Aufruf, 3. Programmzeile. Sie ist nun abgearbeitet.

      Logo fährt mit der Bearbeitung des 2. Aufrufs von HOPPLA in der 4. Programmzeile fort. Die Eingabe beim 2. Aufruf war AUS.
      Also druckt Logo A aus.
      Damit ist der 2. Aufruf von HOPPLA abgearbeitet.
      Logo macht an der Stelle weiter, von der aus der 2. Aufruf erfolgte. Dies war der 1. Aufruf, 3. Programmzeile. Sie ist nun abgearbeitet.

   Logo fährt nun mit der Bearbeitung des 1. Aufrufs von HOPPLA in der 4. Programmzeile fort. Die Eingabe beim 1. Aufruf war HAUS.
   Also druckt Logo H aus.
   Damit ist der 1. Aufruf von HOPPLA abgearbeitet.

Logo macht an der Stelle weiter, von der aus der 1. Aufruf erfolgte.

Abbildung 5.3 zeigt das Diagramm für den Programmfluß.

Die Prozeduren BUCHSTABEN und HOPPLA sind zwar zur Illustration der Rekursion gut geeignet, wenn man jedoch mit den Ergebnissen dieser Prozeduren weiterarbeiten will, ist es besser die Buchstaben nicht auszudrucken sondern in einer Liste abzuspeichern und diese Liste als Funktionswert zurückzugeben. Es wäre also besser, diese Prozeduren als echte Funktionen zu schreiben.

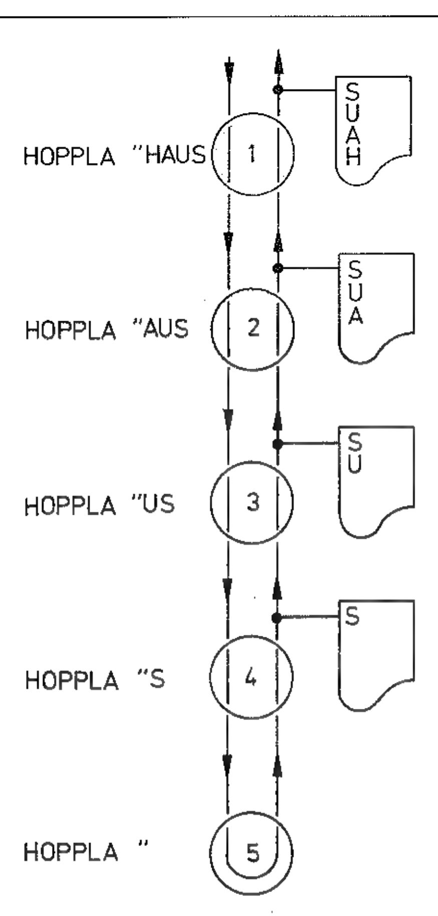


Abbildung 5.3

Der eigentliche Anwendungsbereich für rekursive Prozeduren, die keine Funktionen sind, ist die Graphik. Aus diesem Bereich sollen nun einige Beipiele behandelt werden.

5.6.2 Rekursive Graphik-Prozeduren

5.6.2.1 Zufallsbewegungen

Die nachfolgende Prozedur kann als ein Modell für die Zufallsbewegung (**Brownsche Bewegung**) eines Teilchens angesehen werden, das sich in der Ebene mit konstanter Geschwindigkeit bewegt, aber laufend seine Richtung ändert.

```
PR ZUFALLSBEWEGUNG
 VORWAERTS 5
 RECHTS ZUFALLSZAHL 361
 ZUFALLSBEWEGUNG
ENDE
```



Abbildung 5.4: Zufallsbewegung

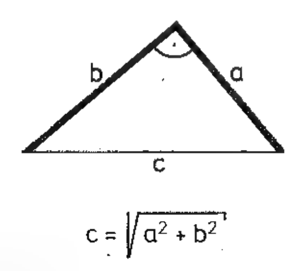
Falls jeweils auch noch die Geschwindigkeit in zufälliger Weise etwa im Bereiche zwischen 0 und 9 Schritten pro Zeiteinheit schwanken soll, so lautet die entsprechende Prozedur:

```
PR ZUFALLSBEWEGUNG
 VORWAERTS ZUFALLSZAHL 10
 RECHTS ZUFALLSZAHL 361
 ZUFALLSBEWEGUNG
ENDE
```

Aufgabe 5.1: Ergänzen Sie die Prozedur ZUFALLSBEWEGUNG so, daß sie nach einer vorgegebenen Anzahl von Schritten gestoppt wird.

5.6.2.2 Die "Wurzelschnecke"

Wie erhält man Wurzel 7 ohne zu rechnen? Mit der Wurzelschnecke. Denn nach dem **Satz des Pythagoras** gilt folgendes:



$$c = \sqrt{a^2 + b^2}$$

Abbildung 5.5: Satz des Pythagoras

Man kann sich die Wurzelschnecke als eine Folge rechtwinkliger Haken (einer davon ist in der obigen Figur fett gezeichnet) vorstellen. Der Punkt im Scheitel des Hakens habe die Koordinaten (:X / :Y). Dann erhält man den Haken durch die folgende Prozedur.

```
PR HAKEN  :X  :Y
 AUFXY 0  0
 AUFKURS  RICHTUNG  :X  :Y
 AUFXY  :X  :Y
 RECHTS 90
 VORWAERTS  30
                                <-- HAKEN XKO YKO
ENDE
```

Durch VORWAERTS 30 wird die Länge des "freien" Schenkels (willkürlich) auf den Wert 30 festgesetzt.

Will man nun einen Haken an den anderen hängen, so muß man in der (freigelassenen) vorletzten Zeile von HAKEN die Prozedur HAKEN mit den Koordinaten des neuen Standorts aufrufen. Dies wird von dem Aufruf HAKEN XKO YKO geleistet.

Die Aufgabe der Prozedur WURZELSCHNECKE besteht nur noch darin, die Prozedur HAKEN mit geeigneten Anfangsparametern so aufzurufen, daß das Ausgangsdreieck gleichschenklig wird. Dies wird hier dadurch erreicht, daß als zweiter Parameter die Zahl 30 gewählt wird.

```
PR WURZELSCHNECKE
 HAKEN  0  30
ENDE
```

Im Grunde genommen hätten wir die Prozedur WURZELSCHNECKE gar nicht gebraucht, sondern HAKEN mit den Parametern 0 und 30 im Direktausführungsmodus aufrufen können.

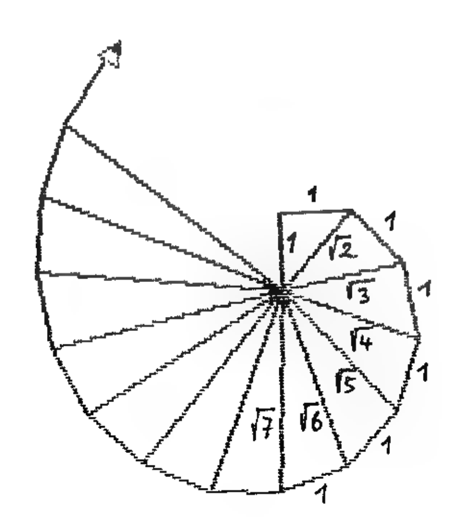


Abbildung 5.6: Wurzelschnecke (Vor Aufruf der Prozedur sollte der vertikale Verzerrungsfaktor mit .SKALA auf den richtigen Wert gesetzt werden; siehe Kapitel 1, Abschnitt 1.3).

Aufgabe 5.2: Ergänzen Sie die Prozedur HAKEN so, daß nur eine bestimmte, vorgegebene Anzahl von Haken gezeichnet wird.

5.6.2.3 Polygonspiralen

In Kapitel 2, Aufgabe 2.5 sollte eine Prozedur POLYGON :N :S (mit Eckenzahl :N und Seitenlänge :S) geschrieben werden. Entsprechend den in Kapitel 2 entwickelten Kenntnissen müßte die Lösung etwa wie folgt ausgesehen haben:

```
PR POLYGON  :N  :S
 WIEDERHOLE  :N  [ VORWAERTS  :S  RECHTS 360  / :N  ]
ENDE
```

In ROSETTE haben wir solche Polygone zusätzlich gedreht. In der folgenden Prozedur wollen wir sie zusätzlich nach dem Drehen noch etwas vergrößern.

```
PR POLYGONSPIRALE  :N  :SEITE  :SEITENZUWACHS  :DREHWINKEL
 POLYGON  :N  :SEITE
 RECHTS  :DREHWINKEL
 POLYGONSPIRALE :N (:SEITE + :SEITENZUWACHS) :SEITENZUWACHS :DREHWINKEL
ENDE
```

In Abbildung 5.7 sind einige Beispiele gegeben.

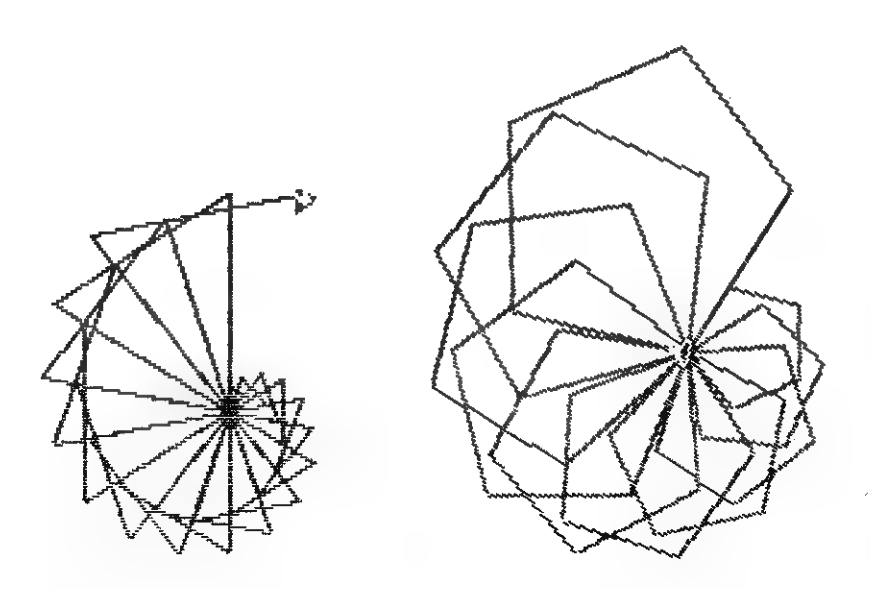


Abbildung 5.7: POLYGONSPIRALE 3 20 5 20 und POLYGONSPIRALE 5 30 5 28 (mit .SKALA 1)

Aufgabe 5.3: Modifizieren Sie die Prozedur POLYGONSPIRALE so, daß sie nach einer bestimmten Anzahl von Schritten beendet wird.

5.6.2.4 Bäume

Die Prozedur BAUM ist typisch für viele rekursive Prozeduren im Graphik-Bereich. Sie ist ein Beispiel für eine verzweigte Rekursion.

```
PR BAUM  :GROESSE
 WENN  :GROESSE < 5  DANN RUECKKEHR
 VORWAERTS  :GROESSE
 LINKS  60
 BAUM  ( :GROESSE * 0.6 )
 RECHTS 120
 BAUM  ( :GROESSE * 0.6 )
 LINKS  60
 RUECKWAERTS  :GROESSE
ENDE
```

Der Aufruf BAUM 60 führt zum folgenden Bild (mit unsichtbarem Igel):

Abbildung 5.8: BAUM 60

Lassen Sie uns jetzt die Prozedur BAUM analysieren. Der Aufruf BAUM 60 erzeugt zwei Aufrufe von BAUM 36 (36 = 60 * 0.6), jeder von ihnen wiederum zwei Aufrufe von BAUM 21.6, jeder davon zwei Aufrufe von BAUM 12.96, jeder davon zwei Aufrufe von BAUM 7.776, und jeder davon schließlich zwei Aufrufe von BAUM 4.6656. Schauen wir uns zunächst diese kleineren Bäume im Direktausführungsmodus (mit verstecktem Igel) an:

(leer)

BAUM 36 BAUM 21.6 BAUM 12.96 BAUM 7.776 BAUM 4.6656

Abbildung 5.9: Die Entstehung von BAUM 60

Wenn man die Wirkungsweise des Aufrufs BAUM 60 verstehen will, so kann man versuchen, ähnlich wie im Protokollmodus jeden einzelnen Schritt nachzuvollziehen. Es kann einem dabei aber leicht wie dem Tausendfüßler gehen, der sich total verhedderte, als er darüber nachdachte, in welcher Reihenfolge er seine Füße bewegen sollte.

Zum Verständnis rekursiver Prozeduren und Funktionen empfiehlt es sich dagegen meist, eine "statische" Sichtweise einzunehmen. Quintessenz dieser statischen Auffassung ist, daß man davon ausgeht, daß die betreffende Prozedur für "kleinere" Parameter genau das Behauptete leistet. (In der Mathematik nimmt man beim Beweisen durch *vollständige Induktion* einen ganz entsprechenden Standpunkt ein). Lassen Sie uns dies einmal für den Aufruf BAUM 60 (mit sichtbarem Igel) durchspielen.

Aufruf: BAUM 60
Der Wert von GROESSE ist nicht gleich 0, also
VORWAERTS 60



Abbildung 5.10.1

LINKS 60



Abbildung 5.10.2

BAUM 36

Abbildung 5.10.3

RECHTS 120

Abbildung 5.10.4

BAUM 36

Abbildung 5.10.5

LINKS 60

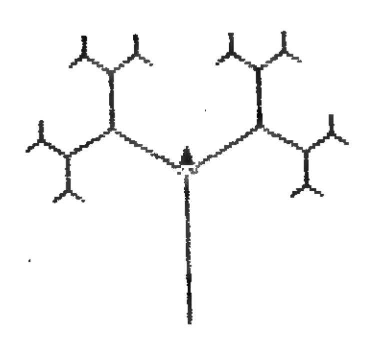


Abbildung 5.10.6

RUECKWAERTS 60

Abbildung 5.10.7

Die Prozedur BAUM zeichnet einen ziemlich "zahmen" Baum.

Aufgabe 5.4: Schreiben Sie Prozeduren zum Zeichnen von stärker "zerzausten" Bäumen.

5.6.3 Rekursive Funktionen

5.6.3.1 Rekursive Funktionen zur Verarbeitung von Zahlen

Die rekursiven Funktionen stellen zweifellos den wichtigsten Aspekt im Themenbereich "Rekursion" dar. Deswegen wird dieses Thema im Kapitel über die vertiefte Behandlung von Funktionen nochmals aufgegriffen. An dieser Stelle sollen nur die Grundlagen zum Verständnis der nachfolgenden weiterführenden Kapitel gelegt werden.

Das Standardbeispiel zur Erläuterung rekursiver Funktionen ist die **Fakultätsfunktion** n! (lies: n Fakultät). Mit n sei im folgenden eine beliebige, natürliche Zahl bezeichnet. Informell läßt sich diese Funktion folgendermaßen beschreiben:

$$n! = 1 * 2 * 3 * 4 * \ldots * (n-1) * n \qquad (5.3)$$

Aber diese Beschreibung ist nicht unproblematisch. Zum Beispiel ist sie in dieser Form für n = 3 nicht korrekt. Besonders die drei Pünktchen verursachen erhebliche Schwierigkeiten wenn man genauer sagen soll, was sie bedeuten.

In der Mathematik hat man eine Technik zur Definition derartiger Funktionen entwickelt. Im Falle der Fakultätsfunktion lautet die Definition:

$$\left.\begin{array}{ll} n! = 1 & \text{falls } n = 1 \text{ ist; und} \\ n! = (n-1)! * n & \text{falls } n \text{ von 1 verschieden ist.} \end{array}\right\} \qquad (5.4)$$

Auf den ersten Blick könnte man meinen, daß hier eine zirkuläre Definition vorliegt, denn das **Definiendum** (Definiendum = das zu Definierende), also die Fakultätsfunktion, kommt nicht nur auf der linken Seite des Gleichheitszeichens sondern auch auf der rechten Seite vor, die üblicherweise dem **Definiens** (Definiens = das Definierende) vorbehalten ist.

Allerdings kommt die Fakultätsfunktion auf der rechten Seite mit einem verkleinerten Argument vor; und das ist die Rettung. Schauen wir uns einmal an, wie dieser Definitionsmechanismus im Detail funktioniert.

Nehmen wir an, jemand will anhand der Definition (5.4) herausfinden, was 4! ist. Das Definitionsschema führt dann zu dem folgenden Prozeß:

```
4!  =  3!  *  4
       3! = 2!  *  3
            2!  =  1!  *  2
                   1!  =  1          also:
            2!  =  1  *  2  =  2     also:
       3!  = 2     *  3   =  6       also:
4!  =   6  *  4   = 24
```

Man nennt dieses Definitionsschema **rekursive Definition** oder auch Definition durch Induktion, denn Rekursion und (vollständige) Induktion hängen aufs engste miteinander zusammen.

Die Logo-Funktion zur Berechnung der Fakultätsfunktion ist nun eine eins-zu-eins Übersetzung des Definitionsschemas (5.4). Wir schreiben FAK :N an Stelle von :N!

```
PR FAK  :N
 WENN  :N = 1  DANN RUECKGABE 1
 RUECKGABE ( FAK  :N-1 )  *  :N
ENDE
```

FAK 4
ERGEBNIS: 24

Schauen wir uns den Ablauf der Funktion im Detail an.

1. Aufruf von FAK mit Eingabe 4.
   Die Eingabe ist nicht gleich 1.
   Also RUECKGABE (FAK 3) * 4; dabei:
   2. Aufruf von FAK mit Eingabe 3
      Die Eingabe ist nicht gleich 1.
      Also RUECKGABE (FAK 2) * 3; dabei:
      3. Aufruf von FAK mit Eingabe 2.
         Die Eingabe ist nicht gleich 1.
         Also RUECKGABE (FAK 1) *2; dabei:
         4. Aufruf von FAK mit Eingabe 1.
            Die Eingabe ist 1.
            Also Rückgabe von 1 an die aufrufende Stelle.
            Dies war der 3. Aufruf von FAK.
         Weiterbearbeitung des 3. Aufrufs:
         Auswertung von (FAK 1) * 2 zu 2.
         Rückgabe von 2 an die aufrufende Stelle.
         Dies war der 2. Aufruf von FAK.
      Weiterbearbeitung des 2. Aufrufs:
      Auswertung von (FAK 2) * 3 zu 6.
      Rückgabe von 6 an die aufrufende Stelle.
      Dies war der 1. Aufruf von FAK.
   Weiterbearbeitung des 1. Aufrufs:
   Auswertung von (FAK 3) * 6 zu 24.
   Rückgabe von 24 an die aufrufende Stelle.

Weiterführung der Arbeit vor dem 1. Aufruf von FAK.

Der Ablaufprozeß dieser Funktion läßt sich ebenfalls durch ein Ablaufdiagramm beschreiben, wie wir es schon in Abschnitt 5.6.2 verwendet haben.

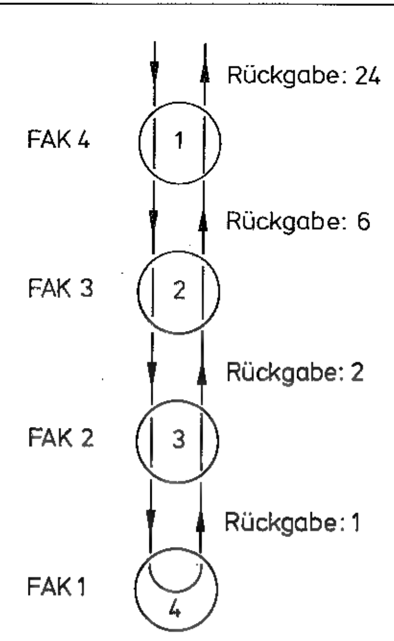


Abbildung 5.11

Ein weiteres typisches Beispiel für rekursive Funktionen ist die Fibonacci-Funktion (nach Leonardo von Pisa, genannt Fibonacci, etwa 1170 - 1240). Die Folge der **Fibonacci-Zahlen** beginnt folgendermaßen:

Stelle:	0	1	2	3	4	5	6	7	8	9	10	11 ...	n ...
Folgenglieder:	0	1	1	2	3	5	8	13	21	34	55	89 ...	FIB(n) ...

Es wird behauptet, daß die Folge aus einer Hypothese zur Vermehrung von Kaninchenpopulationen entstanden ist. Inzwischen gibt es eine Fülle von Deutungen für die Fibonacci-Zahlen. Das Büchlein "Die Fibonaccischen Zahlen" von N.N.Worobjow, Deutscher Verlag der Wissenschaften, Berlin 1971, enthält viel Wissenswertes über diese Zahlen.

Wir sind hier primär am Bildungsgesetz dieser Zahlen interessiert. Ab dem zweiten Glied der Folge ergeben sich die obigen Zahlen nach der Regel: jede Zahl ist die Summe ihrer beiden Vorgänger. Diese Vorschrift läßt sich direkt in einer Logo-Funktion zur Berechnung der Fibonacci-Zahlen umsetzen.

```
PR FIB  :N
 WENN  :N < 2  DANN RUECKGABE :N
 RUECKGABE ( FIB  :N-1 ) + ( FIB  :N-2 )
ENDE
```

FIB 4
ERGEBNIS: 3

Aufgabe 5.5: Billy Bitbeißer hält die runden Klammern in FIB für überflüssig. Probieren Sie seine Version ohne Klammern aus und erklären Sie warum sie nicht funktioniert.

Bei der Auswertung der Fibonacci-Funktion erfolgen (für :N > 1) jeweils zwei rekursive Aufrufe der Funktion. Deshalb fällt das Ablaufdiagramm komplexer aus als im Falle der einfacheren Fakultät.

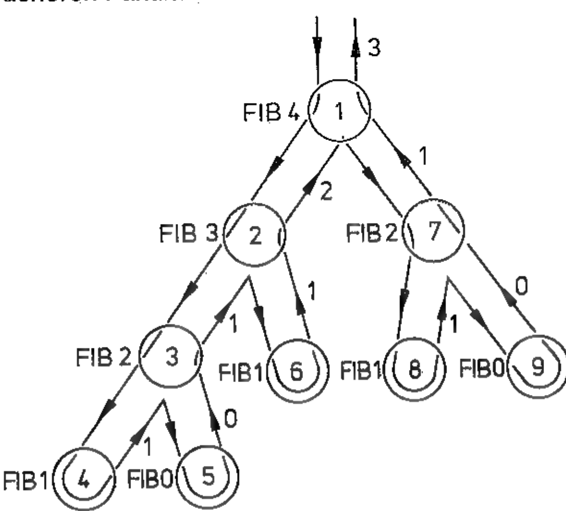


Abbildung 5.12

Als letztes Beispiel aus dem numerischen Bereich wollen wir die Quersumme behandeln.

```
PR QUERSUMME  :X
 WENN  :X  =  "  RUECKGABE 0
 RUECKGABE  ( ERSTES  :X ) + ( QUERSUMME  OHNEERSTES  :X )
ENDE
```

QUERSUMME 125
ERGEBNIS: 8

QUERSUMME "352
ERGEBNIS: 10

Die Operationen ERSTES und OHNEERSTES sind also sowohl auf Zahlen als auch auf Wörter anwendbar.

Aufgabe 5.6: Schreiben Sie Funktionen MIN :L, MAX :L, SUMME :L, MITTELWERT :L, MEDIAN :L, ... welche die durch den Namen angezeigten Größen aus den in der Liste :L enthaltenen Zahlen ermitteln. Die Anzahl der in der Liste :L enthaltenen Zahlen kann dabei unterschiedlich sein und braucht nicht als Parameter eingegeben zu werden.

5.6.3.2 Rekursive Funktionen zur Verarbeitung von Listen und Wörtern

Wenn man mit Listen arbeitet, muß man oft wissen, wie viele Elemente eine Liste entält. In manchen Logo-Versionen gibt es eine Grundfunktion, welche die Elemen-

tezahl von Listen ermittelt. Im deutschen Logo für den Commodore 64 Computer heißt sie LAENGE. In der englischen MIT-Version wird diese Funktion COUNT genannt. Wir wollen sie im folgenden ELEMENTEZAHL nennen. Selbst wenn dies keine Logo-Grundfunktion ist, kann man sie sich leicht selbst schreiben.

```
PR ELEMENTEZAHL  :L
 WENN  :L = [ ]  DANN RUECKGABE 0
 RUECKGABE  1 + ELEMENTEZAHL OHNEERSTES  :L
ENDE
```

ELEMENTEZAHL [A B C D E]
ERGEBNIS: 5

ELEMENTEZAHL [XYZ [U V W] R S]
ERGEBNIS: 4

Wenn man an Stelle der Bedingung :L = [ ] die in Kapitel 4, Abschnitt 4.7 behandelte Funktion LEER? verwendet hätte, würde ELEMENTEZAHL auch für Wörter funktionieren, das heißt die Anzahl der Buchstaben eines Wortes zurückgeben.

Eine weitere nützliche Hilfsfunktion zur Verarbeitung von Listen ist die Funktion PICKE :N :L die das :N-te Objekt aus der Liste :L herauspickt.

```
PR PICKE  :N  :L
 WENN  LEER?  :L  DANN  DRUCKEZEILE  "UNMOEGLICH  AUSSTIEG
 WENN  :N =1  DANN  RUECKGABE  ERSTES  :L
 RUECKGABE  PICKE ( :N-1 )  OHNEERSTES :L
ENDE
```

PICKE 3 [A B C D E]
ERGEBNIS: C

PICKE 2 [X [2 3 4 R X] P Q R]
ERGEBNIS: [2 3 4 R X]

PICKE 5 [W S X]
UNMOEGLICH

Häufig möchte man überprüfen, ob ein bestimmtes Objekt :X in einer Liste :L vorkommt. Wir nennen die entsprechende Prüffunktion ELEMENT?.

```
PR ELEMENT?  :X  :L
 WENN  LEER?  :L  DANN  RUECKGABE  "FALSCH
 WENN  :X  =  ERSTES  :L  DANN  RUECKGABE  "WAHR
 RUECKGABE  ELEMENT?  :X   ( OHNEERSTES  :L )
ENDE
```

ELEMENT? "XY [A 1 XY Z]
ERGEBNIS: WAHR

ELEMENT? "XY [A 1 XYZ]
ERGEBNIS: FALSCH

ELEMENT? [A C] [A B C]
ERGEBNIS: FALSCH

ELEMENT? [A C] [X Y [A C] Z]
ERGEBNIS: WAHR

Aufgabe 5.7: Schreiben Sie eine Funktion ENTHALTEN? :L1 :L2, die überprüft, ob jedes Element der Liste :L1 in der Liste :L2 enthalten ist.

Eine weitere Hilfsfunktion ist ERSETZE :ALT :NEU :L. Sie soll das Objekt :ALT, wo immer es in der Liste :L vorkommt, durch das Objekt :NEU ersetzen.

```
PR ERSETZE :ALT :NEU :L
 WENN LEER? :L DANN RUECKGABE :L
 PRUEFE :ALT = ERSTES :L
 WENNWAHR RUECKGABE MITERSTEM :NEU ________________________
 ________________ ( ERSETZE :ALT :NEU ( OHNEERSTES :L ) )
 RUECKGABE MITERSTEM ( ERSTES :L ) _________________________
 ________________ ( ERSETZE :ALT :NEU ( OHNEERSTES :L ) )
ENDE
```

ERSETZE "A "I [A B R A K A D A B R A]
ERGEBNIS: [I B R I K I D I B R I]

Schauen wir uns noch einmal exemplarisch an, wie ERSETZE arbeitet.

ERSETZE "A "I [A B R A K A D A B R A]

Die Liste :L ist nicht leer. Der Wert von ALT ist A, also gleich dem ersten Element von der Liste :L. Der Funktionswert von ERSETZE ergibt sich also aus dem folgenden Substitutionsprozess:

RUECKGABE MITERSTEM :NEU (ERSETZE :ALT :NEU (OHNEERSTES :L))
= RUECKGABE MITERSTEM "I (ERSETZE "A "I [B R A K A D A B R A]) (*)
= RUECKGABE MITERSTEM "I [B R I K I D I B R I] (* *)
= RUECKGABE [I B R I K I D I B R I]
= [I B R I K I D I B R I]

Der Übergang von der mit (*) zu der mit (* *) gekennzeichneten Zeile zeigt, daß es, ähnlich wie bei der Prozedur BAUM, auch zum Verständnis von rekursiven Funktionen günstig ist, die Dinge statisch zu betrachten. Wenn unsere Funktion ERSETZE richtig arbeitet, und davon wollen wir ja ausgehen, dann sind

ERSETZE "A "I [B R A K A D A B R A] und [B R I K I D I B R I]

schlicht dasselbe.

Betrachten wir nun das nächste Beispiel:

ERSETZE "I "U [M I S [S I S S I] P P I]
ERGEBNIS: [M U S [S I S S I] P P U]

Die obige ERSETZE-Prozedur ersetzt also das Objekt :ALT nur in der Liste :L selbst, nicht jedoch in denjenigen Elementen von :L, die auch wieder Listen sind. Hier (mit den Logo-Abkürzungen) eine verbesserte Version von ERSETZE, die den Ersetzungsprozeß auch auf diejenigen Elemente von :L überträgt, die selbst Listen sind:

```
PR ERSETZE  :ALT  :NEU  :L
 WENN  LEER?  :L   RG  :L
 PRUEFE  LISTE?  ER  :L
 WW  RG  ME ( ERSETZE :ALT :NEU  ER :L ) ( ERSETZE :ALT :NEU  OE :L )
 PRUEFE  :ALT  =  ER  :L
 WW  RG  ME :NEU     ( ERSETZE  :ALT  :NEU  OE  :L )
 RG  ME ( ER  :L )   ( ERSETZE  :ALT  :NEU  OE  :L )
ENDE
```

ERSETZE "I "U [M I S [S I S S I] P P I]
ERGEBNIS: [M U S [S U S S U] P P U]

Das ist es, was wir eigentlich wollten.

Die folgende Funktion dreht die Reihenfolge der Objekte der Liste :L um.

```
PR SPIEGELUNG  :L
 WENN  LEER?  :L  DANN  RUECKGABE  :L
 RUECKGABE  MITLETZTEM  ( ERSTES  :L )  ( SPIEGELUNG  OHNEERSTES  :L )
ENDE
```

Schauen wir uns einmal an, wie SPIEGELUNG auf die Liste [A B C] wirkt.

SPIEGELUNG [A B C]
ERGEBNIS: [C B A]

ERSTES :L ist A. Mit diesem Element als letztem Element wird die SPIEGELUNG von OHNEERSTES :L, also von [B C], bestückt.

Die dritte Zeile der Prozedur Spiegelung durchläuft beim ersten Aufruf von SPIEGELUNG also den folgenden Auswertungsprozeß:

RUECKGABE MITLETZTEM (ERSTES :L) (SPIEGELUNG OHNEERSTES :L)
RUECKGABE MITLETZTEM "A (SPIEGELUNG [B C])
RUECKGABE MITLETZTEM "A [C B]
RUECKGABE [C B A]

Die Prozedur SPIEGELUNG spiegelt jedoch nicht die Unterlisten, die sich eventuell in der Liste :L befinden können. Ein Beispiel:

SPIEGELUNG [A [B C] [D E F]]
ERGEBNIS: [[D E F] [B C] A]

Aufgabe 5.8: Schreiben Sie die Funktion SPIEGELUNG so um, daß auch noch die "inneren" Listen gespiegelt werden. Sie können dazu der zweiten Fassung der ERSETZE-Funktion einige Ideen entnehmen.

Aufgabe 5.9: Schreiben Sie eine Spiegelungsfunktion WORT.SPIEGELUNG, welche die Buchstaben eines Wortes umkehrt.

Ein **Palindrom** ist ein Wort, das von vorn gelesen und von rückwärts gelesen gleich ist. ANNA, IMI und OTTO sind zum Beispiel Palindrome. Auch Zahlwörter können Palindrome sein wie zum Beispiel 100111001. Manche stochastischen (= zufalls-

bedingten) Prozesse bauen auf Palindromen auf. So zum Beispiel der folgende Prozeß: eine Münze habe die Seiten Z (für Zahl) und W (für Wappen). Man werfe die Münze (mindestens zweimal) und schreibe das jeweilige Ergebnis auf, bis ein Palindromwort entstanden ist. Mögliche Stop-Wörter sind WW, ZZ, WZW, ZWZ, WZZW, ZWWZ, WZZZW, u.s.w.. Während des wiederholten Münzwerfens muß jeweils geprüft werden, ob ein vorliegendes Wort ein Palindrom ist oder nicht. Mit Hilfe der Prozedur WORT.SPIEGELUNG können wir leicht ein Prüfwort PALINDROM? schreiben, das genau diese Frage beantwortet.

```
PR PALINDROM?  :W
 RUECKGABE  :W  =  WORT.SPIEGELUNG  :W
ENDE
```

PALINDROM? "OTTO
ERGEBNIS: WAHR

PALINDROM? "EMMA
ERGEBNIS: FALSCH

Die folgende Funktion gibt zufällige, aus 0 und 1 bestehende Palindromwörter zurück.

```
PR PALINDROM
 RUECKGABE  PALINDROM.1  ( WORT ( ZZ 2 ) ( ZZ 2 ) )
ENDE

PR PALINDROM.1  :W
 WENN  PALINDROM?  :W  DANN  RUECKGABE  :W
 RUECKGABE  PALINDROM.1  ( WORT  :W  ZZ 2 )
ENDE
```

Man kann PALINDROM etwa folgendermaßen im interaktiven Betrieb verwenden:

WH 5 [DZ PALINDROM]

101
00
01110
11
1001

Aufgabe 5.10: Schreiben Sie eine "Simulationsumgebung" für das Palindrom-Spiel, mit der Sie das Spiel sehr oft hintereinander ausführen können. Bestimmen Sie die mittlere Länge der Palindrom-Wörter.

Zum Abschluß dieses Abschnitts soll noch die verbesserte Version des eingangs behandelten Buchstaben-Problems gegeben werden.

```
PR BUCHSTABENLISTE  :W
 WENN   LEER?  :W   DANN  RUECKGABE [ ]
 RUECKGABE  MITERSTEM  (ERSTES :W) (BUCHSTABENLISTE OHNEERSTES  :W)
ENDE
```

BUCHSTABENLISTE "HAUS
ERGEBNIS: [H A U S]

Aufgabe 5.11: Schreiben Sie eine Funktion GESPIEGELTE.BUCHSTABENLISTE

5.7 Explizite Auswertung von Listen

Mit Hilfe des TUE Befehls kann man die Abarbeitung (Interpretation) von Listen explizit erzwingen. Dieser Befehl macht Logo zu einem außerordentlich starken Werkzeug; er ermöglicht zum Beispiel solche Dinge, wie:

- die Übergabe von Funktionen oder ganzen Programmen als (aktuelle) Eingabewerte für Prozeduren;
- die Einführung neuer Kontrollstrukturen;
- die Änderung des Interpreterverhaltens von Logo.

Auch in bezug auf diesen Themenkreis sollen in diesem Kapitel nur die Grundlagen gelegt werden; die Vertiefung erfolgt in den nachfolgenden Kapiteln.

Die Prozedur TUE erwartet als Eingabeobjekt eine Liste. TUE interpretiert die Listenelemente als Logo-Befehle und führt sie aus. Ein Beispiel: an Stelle von

DRUCKE "REUTLINGEN

könnte man genausogut schreiben:

TUE [DRUCKE "REUTLINGEN]

Noch ein Beispiel: an Stelle von

8 * SIN 27

könnte man auch schreiben:

TUE [8 * SIN 27]

Das sieht auf den ersten Blick vielleicht noch nicht sehr aufregend aus. Schauen wir uns an, was man damit machen kann.

```
PR INTERAKTIVE.FUNKTIONSAUSWERTUNG
 DRUCKE  [ GEBEN SIE EINEN FUNKTIONSTERM EIN: ]
 DRUCKEZEILE  TUE EINGABE
 INTERAKTIVE.FUNKTIONSAUSWERTUNG
ENDE
```

Einige Beispiele:

INTERAKTIVE.FUNKTIONSAUSWERTUNG
GEBEN SIE EINEN FUNKTIONSTERM EIN: 8 * SIN 27
3.36192
GEBEN SIE EINEN FUNKTIONSTERM EIN: (QW 5) * (COS 60)
1.11803
GEBEN SIE EINEN FUNKTIONSTERM EIN: 2*3+5
11

GEBEN SIE EINEN FUNKTIONSTERM EIN: ERSTES "HAUS
H
GEBEN SIE EINEN FUNKTIONSTERM EIN: SPIEGELUNG [1 2 3 4]
[4 3 2 1]

Da sich diese Auswertungsprozedur immer wieder selbst aufruft, kommt man aus diesem Zyklus nur durch den Prozedurabbruch Ctrl-G (oder durch das Provozieren einer Fehlermeldung) heraus.

Aufgabe 5.12: Modifizieren Sie die obige Prozedur so, daß im Dialog entschieden wird, ob die Prozedur beendet oder nochmals aufgerufen werden soll.

Wir wollen dieses Beispiel im folgenden noch so ausbauen, daß auch Funktionsterme mit Variablen bearbeitet werden können.

```
PR INTERAKTIVE.FUNKTIONSAUSWERTUNG.MIT.EINER.VARIABLEN
 DRUCKE [ GEBEN SIE EINEN FUNKTIONSTERM IN DER VARIABLEN X EIN: ]
 SETZE "L1 EINGABE
 DRUCKE [ WELCHEN WERT SOLL X HABEN? ]
 SETZE "X ERSTES EINGABE
 DRUCKEZEILE TUE :L1
 INTERAKTIVE.FUNKTIONSAUSWERTUNG.MIT.EINER.VARIABLEN
ENDE
```

Einige Beispiele (die "Antworten" des Computers sind der Deutlichkeit halber kursiv wiedergegeben):

INTERAKTIVE.FUNKTIONSAUSWERTUNG.MIT.EINER.VARIABLEN
GEBEN SIE EINEN FUNKTIONSTERM IN DER VARIABLEN X EIN: (QW :X) * (SIN :X)
WELCHEN WERT SOLL DIE VARIABLE X HABEN? 50
5.41675
GEBEN SIE EINEN FUNKTIONSTERM IN DER VARIABLEN X EIN: :X * :X * :X
WELCHEN WERT SOLL DIE VARIABLE X HABEN? 2
8

Der TUE-Befehl hat es also möglich gemacht, daß ganze Funktionsterme (und nicht nur solche arithmetischer Natur) im Dialog eingelesen und dann ausgewertet werden konnten. Ebenso ist es möglich, ganze Funktionen als Parameter an Prozeduren und Funktionen zu übergeben. Dies ist zum Beispiel in der Mathematik im Zusammenhang mit der Integration und Differentiation von Funktionen wichtig.

Damit nicht genug. Ebenso wie Funktionen können Prozeduren, also Programme, als Parameter übergeben werden. Es gibt nur sehr wenige Programmiersprachen, in denen derartige Dinge möglich sind.

An dieser Stelle soll abschließend noch ein Versprechen aus Kapitel 2, Abschnitt 2.4.2 eingelöst werden. Wir werden sehen, wie man sich mit Hilfe des TUE-Befehls einen privaten Funktions-Interpreter schreiben und dadurch die Oberfläche von Logo, also die Art und Weise, in der Logo mit dem Benutzer kommuniziert, verändern kann.

Nehmen wir an, Sie arbeiten mit der englischen LCSI-Version von Logo, die bei Rückgabe von nicht weiterverarbeiteten Funktionswerten eine Meldung der Art

YOU DIDN'T TELL ME WHAT TO DO WITH ... oder
YOU DON'T SAY WHAT TO DO WITH ... oder
I DON'T KNOW WHAT TO DO WITH ... (*)

bringt. Vielleicht gefällt Ihnen die Reaktion der MIT-Version besser, die in diesem Falle im Direktausführungsmodus einfach:

ERGEBNIS: ...

ausdruckt. Dann können Sie durch die folgende kleine Prozedur erreichen, daß dies auch im LCSI Logo geschieht. Damit Sie im Dialogbetrieb erkennen, daß Sie sich im Augenblick in Ihrem privaten Auswertungszyklus befinden, wird an Stelle des üblichen Bereitschaftszeichens das Zeichen # ausgedruckt.

```
TO PRIVATE.FUNCTION.EVALUATOR
 TYPE  "#
 ( PRINT  "ERGEBNIS:  RUN READLIST )
 PRIVATE.FUNCTION.EVALUATOR
END
```

TYPE entspricht DRUCKE, PRINT entspricht DRUCKEZEILE, RUN entspricht TUE und READLIST entspricht dem EINGABE-Befehl. Ähnlich wie in dem Programm zur interaktiven Funktionsauswertung wird der Auswertungszyklus dieser Prozedur unterbrochen, wenn Ctrl-G eingegeben wird, wenn eine Fehlermeldung verursacht wird oder wenn die Auswertung der Eingabeliste nicht in einem Funktionswert resultiert.

(*) Hierzu das folgende kleine Zitat aus dem Logo-Buch von Michael Friendly aus Downsview, Ontario (Kanada): ... When that happens, I can't help muttering, "What do you think you should do with it? Why don't you just print it." ...

Kapitel 6: Funktionen und Prozeduren in vertiefter Behandlung

6.1 Lokale Variable

Lassen Sie uns die beiden folgenden Prozeduren im Detail studieren.

```
PR DEMO  :X
 DRUCKEZEILE  :X
 ERHOEHUNG  :X
 DRUCKEZEILE  :X
ENDE

PR ERHOEHUNG  :X
 SETZE  "X  :X+1
 DRUCKEZEILE  :X
ENDE
```

Der Aufruf DEMO 3.14 führt zu folgendem Ausdruck:

3.14
4.14
3.14

Wie ist dies erklärbar? Der Aufruf DEMO 3.14 bewirkt zunächst den ersten Ausdruck der Zahl 3.14. Danach wird die Prozedur ERHOEHUNG mit dem aktuellen Parameter 3.14 aufgerufen. Dieser Wert wird zunächst um Eins erhöht, und dieser erhöhte Wert (4.14) wird innerhalb der Prozedur ERHOEHUNG ausgedruckt. Nach Abarbeitung der Prozedur ERHOEHUNG kehrt Logo wieder zur Prozedur DEMO zurück, um die dritte Zeile dieser Prozedur auszuführen.

Jetzt kommt der entscheidende Moment. In der Prozedur ERHOEHUNG war der Wert der Variablen X um Eins erhöht worden. Dennoch wird in der dritten Zeile von DEMO nicht 4.14 sondern 3.14 ausgedruckt. Die Prozedur DEMO hat also nicht mit dem X-Wert der Prozedur ERHOEHUNG sondern mit ihrem eigenen X-Wert, wie er vor Aufruf der Prozedur ERHOEHUNG vorlag, weitergearbeitet.

Dieses Beispiel verdeutlicht die Wirkungsweise von lokalen Variablen. Nehmen wir an, FOO sei eine beliebige Prozedur. Eine Variable X heißt **lokal** relativ zu FOO (oder: lokale Variable von FOO), wenn man nur von FOO aus, nicht aber von außerhalb, Zugriff auf X hat. Jede Veränderung der lokalen Variablen X ist nur innerhalb von FOO wirksam. Die Variable X existiert nur, solange die Prozedur FOO läuft. (An Stelle von FOO und X kann natürlich jeder beliebige andere Prozedur- beziehungsweise Variablenname verwendet werden).

Es gibt verschiedene Methoden, um lokale Variable zu erzeugen. Zunächst einmal sind alle Variablen, die als (formale) Eingabeparameter einer Prozedur vorkommen, lokal in bezug auf diese Prozedur. Im obigen Beispiel ist die Variable X lokal sowohl in DEMO als auch in ERHOEHUNG.

In den meisten Logo-Versionen gibt es darüber hinaus noch die Möglichkeit, lokale

Variable während des Ablaufs von Prozeduren explizit durch den Befehl LOCAL "X (deutsch: LOKAL "X) zu erzeugen. Im LCSI Logo kann man auf einen Schlag mehrere lokale Variable generieren: durch den Befehl LOCAL [A B C D] werden zum Beispiel die lokalen Variablen A, B, C und D erzeugt. Natürlich funktioniert auch der Befehl LOCAL :L, wenn :L eine Liste der Form [X Y Z] ist. (Im Zusammenhang mit der Syntaxanylyse von Listen sei an dieser Stelle nochmals auf den Abschnitt 4.3.2.1 hingewiesen).

Hierzu als Beispiel ein Programm zur Aufstellung von *Tilgungsplänen*. (Im Hinblick auf die vertiefte mathematische Analyse des Problems wird auf das im Schöningh Verlag, Paderborn, erschienene Buch "Dynamische Prozesse und ihre Mathematisierung durch Differenzengleichungen" der Autoren R.Dürr/J.Ziegenbalg verwiesen).

```
PR TILGUNGSPLAN  :ANFANGSDARLEHEN  :ZINSSATZ  :ANNUITAET
 LOKAL  "LAUFZEIT
 SETZE  "LAUFZEIT 0
 LOKAL  "RESTSCHULD
 SETZE  "RESTSCHULD  :ANFANGSDARLEHEN
 LOKAL  "Q
 SETZE  "Q  ( 1 +  :ZINSSATZ / 100 )
 SOLANGE  [ :RESTSCHULD > 0 ] ____________________
 _______  [ DRUCKE  :LAUFZEIT ____________________
 __________ DRUCKE LEERWORT ______________________
 __________ DRUCKEZEILE  :RESTSCHULD _____________
 __________ SETZE  "LAUFZEIT  :LAUFZEIT + 1 ______
 __________ SETZE  "RESTSCHULD  ( :RESTSCHULD * :Q - :ANNUITAET ) ]
ENDE
```

Die lokalen (Hilfs-) Variablen LAUFZEIT und RESTSCHULD und Q existieren nur, solange die Prozedur TILGUNGSPLAN läuft. Mit Ablauf dieser Prozedur werden sie gelöscht. Im englischsprachigen LCSI Logo hätte man sie auch durch den Aufruf LOCAL [LAUFZEIT RESTSCHULD Q] erzeugen können.

In der Prozedur TILGUNGSPLAN wurde noch die Hilfsprozedur LEERWORT aus Kapitel 4, Abschnitt 4.2.4 verwendet.

Sprachelemente wie SOLANGE bezeichnet man in der Informatik als *Kontrollstrukturen*. Mit ihnen läßt sich der Ablauf von Programmen steuern. SOLANGE ist zwar kein Grundwort von Logo, man kann sich diese Prozedur aber sehr leicht selber schreiben. In Kapitel 7 werden wir im Detail auf die Frage eingehen, wie man solche Kontrollstrukturen in Logo selbst erstellen kann. Für den Augenblick genügt es, die SOLANGE-Prozedur aus Abschnitt 7.1 wie eine gewöhnliche Prozedur einzugeben und zu definieren. Man kann die SOLANGE-Prozedur wie jede andere Prozedur auch auf Diskette speichern und jedesmal, wenn man sie benötigt, in den Speicher laden. Es ist dann fast so als ob SOLANGE ein Logo-Grundwort ist, von dem man nur wissen muß, daß es zwei Eingabeparameter hat: eine Liste, in der eine Bedingung steht und eine zweite Liste, welche die Handlung enthält, die auszuführen ist, solange die Bedingung erfüllt ist.

Lassen Sie uns die Prozedur TILGUNGSPLAN konkret auf die Tilgung von Bausparverträgen anwenden. Ein **Bausparvertrag** wird nach einer Ansparphase zugeteilt, wenn etwa ein Drittel der Vertragssumme angespart ist. Bei einer Vertragssumme von 100 000 DM beträgt das Anfangsdarlehen also etwa 67 000 DM. Bei Bausparkonditionen, die man in der Praxis durchaus vorfinden kann (oder konnte), beläuft sich die Annuität auf 6000 DM bei einem Zinssatz von 4.5 %.

Die Tilgung eines solchen Bausparvertrages kann nun mit Hilfe des obigen Programmes durch den Aufruf TILGUNGSPLAN 67000 4.5 6000 simuliert werden; (Zinssatz in Hundertsteln; siehe: allgemeine Hinweise zu Beginn des Buches). Wir erhalten (abgesehen möglicherweise von der spaltengerechten Darstellung) den Ausdruck

```
 0   67000.00
 1   64014.90
 2   60895.60
 3   57635.90
 4   54229.50
 5   50669.80
 6   46949.90
 7   43062.60
 8   39000.50
 9   34755.50
10   30319.50
11   25683.80
12   20839.60
13   15777.40
14   10487.30
15    4959.28
```

Nach diesem Stand wird die Restschuld durch eine Schlußzahlung in entsprechender Höhe annulliert.

Aufgabe 6.1: Im obigen Programm TILGUNGSPLAN wird von einer einmaligen jährlichen Zahlung von 6000 DM ausgegangen. Schreiben Sie das Programm für die folgenden (festbleibenden) Tilgungsraten um:
(a) 1500 DM pro Vierteljahr;
(b) 500 DM pro Monat.
Vergleichen Sie die daraus resultierenden Tilgungsprozesse.
(SOLANGE kann aus Abschnitt 7.1 übernommen werden).

Aufgabe 6.2: (a) Löschen Sie alles, was nicht zum Lauf der Prozedur TILGUNGSPLAN erforderlich ist.
(b) Bauen Sie in der Prozedur TILGUNGSPLAN unmittelbar vor dem Wort SOLANGE den Befehl ZEIGE NAMEN ein und beobachten sie seine Wirkung.
(c) Verfahren Sie entsprechend mit allen anderen Prozeduren dieses Kapitels.

Variable bestehen stets aus zwei Bestandteilen: ihrem Namen und ihrem Wert. Man kann sie sich als in einer Tabelle angeordnet vorstellen. Hier eine Phantasietabelle, wie sie im Laufe des Arbeitens mit Logo zu einem bestimmten Zeitpunkt entstanden sein könnte:

Variablenname	Wert
X	3.14
Y	17.56
APFEL	APPLE
HAUS	HOUSE
RESTSCHULD	4959.28
ZINSSATZ	4.5
...	...

Diese Tabelle wird gelegentlich auch als Bibliothek bezeichnet. In Logo gibt es nun sehr viele Bibliotheken. Zu jeder Prozedur gehört eine **Privatbibliothek**, in der ihre lokalen Variablen gespeichert sind und zu der nur sie selbst Zugang hat.

Variable, die zu keiner der definierten Prozeduren lokal sind, heißen **global**. Dazu gehören zum Beispiel alle Variablen, die im Direktausführungmodus durch das SETZE-Kommando erzeugt werden. Alle globalen Variablen sind ebenfalls in einer Bibliothek zusammengefaßt. Wenn man im Direktausführungsmodus den Befehl ZEIGE NAMEN eingibt, listet Logo genau die zu diesem Zeitpunkt existierenden globalen Variablen mit ihren Werten auf. Der Befehl ZEIGE NAMEN kann auch innerhalb von Prozeduren verwendet werden. Wenn er zum Beispiel in einer Prozedur BAR (ein Phantasiename) aufgerufen wird, so werden alle Namen aufgelistet, die zum Zeitpunkt des Aufrufs bekannt sind, also insbesondere auch die lokalen Variablen von BAR.

Lassen Sie uns nun noch einmal Rückschau halten auf das eingangs gegebene Beispiel. Der Eindeutigkeit halber wollen wir annehmen, daß vor der Definition von DEMO ADE eingegeben wurde, daß wir also auf einer "tabula rasa" beginnen. Im Direktausführungsmodus (auch als "top level" bezeichnet) hat man nur Zugang zur Bibliothek der globalen Variablen, die in diesem Fall aber leer ist.

Bibliothek der globalen Variablen

Variablenname	Wert
leer	leer

Durch den Aufruf von DEMO mit dem formalen Parameter X, dessen aktueller Wert 3.14 ist, wird die folgende Privatbibliothek von DEMO aufgebaut:

Privatbibliothek von DEMO

Variablenname	Wert
X	3.14

Dieser Wert (3.14) wird als erster Wert ausgedruckt.

Beim Aufruf von ERHOEHUNG wird nun die folgende Bibliothek aufgebaut:

Priviatbibliothek von ERHOEHUNG

Variablenname	Wert
X	3.14

Während des Ablaufs von ERHOEHUNG wird der Wert von X in der Privatbibliothek von ERHOEHUNG modifiziert:

Privatbibliothek von ERHOEHUNG

Variablenname	Wert
X	~~3.14~~ 4.14

Dieser Wert (4.14) wird als zweiter Wert ausgedruckt.

Die Privatbibliothek von DEMO bleibt von der Erhöhung jedoch völlig unberührt. Wenn Logo nach Ablauf der Prozedur ERHOEHUNG also zu DEMO zurückkehrt, wird die Variable X in der Bibliothek von DEMO gesucht und hat somit wieder den ursprünglichen Wert. Dieser Wert wird nochmals, jetzt als dritter Wert der Variablen X, ausgedruckt.

Das Arbeiten mit lokalen Variablen dient folgenden Zielsetzungen:

(1.) Der Vermeidung von Kollisionen bei der Vergabe von Variablennamen. Stellen Sie sich vor, es gäbe nur globale Variable und man sei dabei, ein großes Programmpaket zu erstellen. Dann könnte es vorkommen, daß in verschiedenen Prozeduren, an denen möglicherweise verschiedene Personen arbeiten, dieselben Variablennamen verwendet werden. Die Veränderung eines Variablenwertes in jeder der Prozeduren hätte dann Konsequenzen für jede der anderen Prozeduren. Die Gefahr eines heillosen Durcheinanders wäre nur sehr schwer zu vermeiden. Wer einmal ein großes BASIC-Programmpaket geschrieben hat, wird diese Erfahrung gemacht haben. (In BASIC gibt es nur globale Variable, deren Namen zudem meist nur aus maximal zwei signifikanten Zeichen bestehen).
Das eingangs gegebene Beispiel zeigt, daß derartige Namenskonflikte durch die Verwendung lokaler Variablen vermieden werden können.

(2.) Lokale Variable unterstützen das modulare Programmieren. Die Interaktion jedes Moduls (das heißt jeder Prozedur) mit der "Außenwelt" sollte möglichst ohne Nebenwirkungen und nur über exakt definierte Schnittstellen vonstatten gehen. Die Schnittstelle beim Eintritt in den Modul ist durch die Parameterliste der Prozedur, die Schnittstelle beim Austritt aus dem Modul durch die Funktionswertrückgabe gegeben. (Auch diese Interpretation spricht für die Methode des *funktionalen* Programmierens).

Bei der Verwendung globaler Variablen in Prozeduren ergeben sich häufig unerwünschte und schlecht kontrollierbare Seiteneffekte. Die extensive Verwendung von globalen Variablen stellt deshalb einen Verstoß gegen die modulare Vorgehensweise dar. Man sollte nur in Ausnahmefällen davon Gebrauch machen. Näheres dazu im nächsten Abschnitt über kontextgebundenes Programmieren.

6.2 Methoden der Variablenbindung / kontextgebundenes Programmieren / freie Variable

Der Vorgang des Aufsuchens von Variablen in den verschiedenen Bibliotheken und die Belegung mit den jeweiligen Werten wird als Variablenbindung (kurz: Bindung) bezeichnet. In der englischsprachigen Literatur werden die Begriffe binding bzw. scoping (scope: Reichweite, Gültigkeitsbereich) verwendet. Logo praktiziert das Verfahren der sogenannten **dynamischen Variablenbindung** (englisch: dynamic binding oder dyamic scoping). Bei dieser Methode ergibt sich der Wert einer (freien) Variablen grob gesprochen aus der "Laufzeitumgebung" der Prozedur, in der die betreffende Variable vorkommt. Es läuft nach den folgenden Regeln ab: kommt in einer Prozedur (wir nennen sie zum Beispiel P1) ein Variablenname (etwa XYZ) vor, so durchsucht Logo zunächst die Privatbibliothek von P1. Wenn Logo den Namen findet, so weist es der Variablen den zugehörigen Wert zu. Andernfalls prüft Logo, ob der Variablenname in der Prozedur vorkommt, von der aus P1 aufgerufen wurde. Nehmen wir an, dies sei die Prozedur P2. Kommt XYZ in der Privatbibliothek von P2 vor, so belegt Logo die Variable mit dem Wert aus dieser Bibliothek. Andernfalls prüft Logo, ob der Variablenname in der Prozedur vorkommt, von der aus P2 aufgerufen wurde, u.s.w.. Ist XYZ in keiner dieser privaten Bibliotheken zu finden, so prüft Logo schließlich, ob die Variable in der Bibliothek der globalen Variablen enthalten ist. Wenn dies der Fall ist, so weist Logo der Variablen den entsprechenden Wert zu; andernfalls meldet es sich mit der Fehlermeldung "NAME XYZ UNBEKANNT". Eine entsprechende Reihenfolge gilt auch bei der Bestückung von Variablen mit Hilfe des SETZE-Befehls.

Aufgabe 6.3: Entscheiden Sie, welche der im folgenden Beispiel vorkommenden Variablen lokal und welche global sind.

ADE

SETZE "Y 5.26

Funktionsdefinition:

```
PR FOO  :X
 DRUCKE  :X
 SETZE  "Z  0
 DRUCKEZEILE  :Z
ENDE
```

FOO 12

Funktionsdefinition:

```
PR BAR  :R
 DRUCKE  :R
 SETZE  "R  17
 DRUCKEZEILE  :R
ENDE
```

BAR 2.5

Überzeugen Sie sich von der Richtigkeit Ihrer Lösung, indem Sie jetzt das Kommando ZEIGE NAMEN eingeben.

Die oben abstrakt beschriebenen Bindungs-Regeln von Logo sollen nun durch konkrete Beispiele untermauert werden. Die folgenden Funktionen erklären sich von selbst.

```
PR MEHRWERTSTEUER  :NETTOPREIS  :MEHRWERTSTEUERSATZ
 RUECKGABE  (  :NETTOPREIS  *  :MEHRWERTSTEUERSATZ / 100 )
ENDE

PR BRUTTOPREIS  :NETTOPREIS  :MEHRWERTSTEUERSATZ
 RUECKGABE (  :NETTOPREIS + ______________________________
 _______  MEHRWERTSTEUER :NETTOPREIS :MEHRWERTSTEUERSATZ )
ENDE
```

MEHRWERTSTEUER 500 14 (Mehrwertsteuersatz in Hundertsteln)
ERGEBNIS: 70.0

BRUTTOPREIS 200 14
ERGEBNIS: 228.0

Wir können diese Funktionen zum Beispiel in einer Prozedur wie RECHNUNG verwenden:

```
PR RECHNUNG  :NETTOPREIS.1  :NETTOPREIS.2  :MEHRWERTSTEUERSATZ
 DRUCKEZEILE  "RECHNUNG
 ( DRUCKEZEILE [ 1. POSTEN: ]  :NETTOPREIS.1 )
 ( DRUCKEZEILE [ 2. POSTEN: ]  :NETTOPREIS.2 )
 DRUCKE  "MEHRWERTSTEUER:
 DRUCKEZEILE  MEHRWERTSTEUER ( :NETTOPREIS.1  +  :NETTOPREIS.2 ) ______
 _______________________  :MEHRWERTSTEUERSATZ
 DRUCKE  "BRUTTOPREIS:
 DRUCKEZEILE  BRUTTOPREIS  ( :NETTOPREIS.1  +  :NETTOPREIS.2 ) ______
 _____________________  :MEHRWERTSTEUERSATZ
ENDE
```

Der Aufruf RECHNUNG 250 300 14 ergibt den folgenden Ausdruck.

RECHNUNG
1. POSTEN: 250.0
2. POSTEN: 300.0
MEHRWERTSTEUER: 77.0
BRUTTOPREIS: 627.0

Aufgabe 6.4: Erweitern Sie die RECHNUNGS-Prozedur im Hinblick auf die Sachlogik des Problems, und verschönern Sie insbesondere den Rechnungs-Ausdruck durch "Kosmetik". (Hierfür erweisen sich die Prozeduren der "Drucke-stellengerecht-Umgebung" aus Abschnitt 9.3 als besonders nützlich).

In derartigen Beispielen wird es manchmal als lästig empfunden, wenn man immer alle Variablen als Parameter übergeben muß. Im obigen Fall betrifft dies besonders die Variable MEHRWERTSTEUERSATZ; denn der Wert dieser Variablen ändert sich während des gesamten Ablaufs der RECHNUNGS-Prozedur nicht. Der Mehrwertsteuersatz ist im Kontext der RECHNUNGS-Prozedur konstant.

Das von Logo praktizierte Verfahren der dynamischen Variablenbindung ermöglicht aber auch die folgende "kontextgebundene" Lösung des Rechnungsschreibungs-Problems. (Der an die Prozedurnamen angehängte Zusatz V2 soll für "Ver-

sion 2" stehen. Wir werden im nächsten Abschnitt noch weitere Versionen des Funktionenpaares MEHRWERTSTEUER / BRUTTOPREIS betrachten, die dann mit V3 und V4 für "Version 3" bzw. "Version 4" bezeichnet werden).

```
PR MEHRWERTSTEUER.V2  :NETTOPREIS
 RUECKGABE  ( :NETTOPREIS  *  :MEHRWERTSTEUERSATZ / 100 )
ENDE

PR BRUTTOPREIS.V2  :NETTOPREIS
 RUECKGABE ( :NETTOPREIS  +  MEHRWERTSTEUER.V2  :NETTOPREIS )
ENDE

PR RECHNUNG.V2  :NETTOPREIS.1  :NETTOPREIS.2  :MEHRWERTSTEUERSATZ
 DRUCKEZEILE  "RECHNUNG
 ( DRUCKEZEILE  [ 1. POSTEN: ]  :NETTOPREIS.1 )
 ( DRUCKEZEILE  [ 2. POSTEN: ]  :NETTOPREIS.2 )
 DRUCKE  "MEHRWERTSTEUER:
 DRUCKEZEILE  MEHRWERTSTEUER.V2  ( :NETTOPREIS.1  +  :NETTOPREIS.2 )
 DRUCKE  "BRUTTOPREIS:
 DRUCKEZEILE  BRUTTOPREIS.V2  ( :NETTOPREIS.1  +  :NETTOPREIS.2 )
ENDE
```

Wie in der eingangs gegebenen Version führt der Aufruf RECHNUNG.V2 250 300 14 zu dem folgenden Ausdruck:

RECHNUNG
1. POSTEN: 250.0
2. POSTEN: 300.0
MEHRWERTSTEUER: 77.0 (*)
BRUTTOPREIS: 627.0 (* *)

Die Variable MEHRWERTSTEUERSATZ ist nun nicht mehr lokal in der Funktion MEHRWERTSTEUER.V2. Man sagt, daß sie in dieser Funktion als **freie Variable** vorkommt. Beim Lauf der Funktion MEHRWERTSTEUER.V2 muß der Wert dieser freien Variablen ermittelt werden. Logo schaut zunächst in der Privatbibliothek von MEHRWERTSTEUER.V2 nach. Dort findet es die Variable MEHRWERTSTEUERSATZ aber nicht.

Untersuchen wir zunächst den Fall der oben durch (*) gekennzeichneten Zeile. Die Funktion MEHRWERTSTEUER.V2 wurde beim Ausdrucken dieser Zeile von der Prozedur RECHNUNG.V2 aus aufgerufen. Also versucht Logo, den Wert der Variablen MEHRWERTSTEUERSATZ in der Privatbibliothek von RECHNUNG.V2 zu finden. Da MEHRWERTSTEUERSATZ als lokale Variable in RECHNUNG.V2 vorkommt, gelingt dieser Belegungsversuch. Der Funktionswert von MEHRWERTSTEUER.V2 kann also ermittelt und anschließend ausgedruckt werden.

Nun zu der durch (* *) gekennzeichneten Zeile. Die Prozedur MEHRWERTSTEUER.V2 wurde in diesem Fall von BRUTTOPREIS.V2 aus aufgerufen. Logo versucht also, den Wert von MEHRWERTSTEUERSATZ in der Privatbibliothek der Funktion BRUTTOPREIS.V2 zu ermitteln. Aber auch dort ist keine Variable unter diesem Namen zu finden. Die Funktion BRUTTOPREIS.V2 wurde von RECHNUNG.V2 auf-

gerufen. Also sucht Logo in der Privatbibliothek von RECHNUNG.V2 weiter nach dem Wert von MEHRWERTSTEUERSATZ. Dort ist er dann auch tatsächlich zu finden. Damit kann der Funktionswert von MEHRWERTSTEUER.V2 und anschließend der von BRUTTOPREIS.V2 ermittelt und ausgedruckt werden.

Diese zweiten Versionen des MEHRWERTSTEUER.V2 / BRUTTOPREIS.V2 - Funktionenpaares wurden als kontextgebunden bezeichnet, weil sie nur im Kontext einer übergeordneten Prozedur (hier RECHNUNG.V2) lauffähig sind.

Hätte man die Prozeduren MEHRWERTSTEUER.V2 und BRUTTOBETRAG.V2 im Direktausführungsmodus aufgerufen, so wäre der Name MEHRWERTSTEUERSATZ nicht definiert gewesen. Man könnte dem dadurch Abhilfe schaffen, daß man eine globale Variable MEHRWERTSTEUERSATZ definiert; etwa durch:

SETZE "MEHRWERTSTEUERSATZ 14

Eine noch bessere Möglichkeit, den Mehrwertsteuersatz bereitzustellen ist die Definition einer entsprechenden konstanten Funktion:

```
PR MEHRWERTSTEUERSATZ
 RUECKGABE  14
ENDE
```

Aufgabe 6.5: Schreiben Sie die Prozedur RECHNUNG so um, daß der Mehrwertsteuersatz in der Form einer konstanten Funktion und nicht in der Form einer Variablen verwendet wird.

6.2.1 Eine Detailstudie zum Verfahren der dynamischen Variablenbindung

Die Methode der dynamischen Variablenbindung ist ein sehr flexibles Verfahren, besonders in Hinblick auf die Belegung und Abfrage von Variablen, die erst zur Laufzeit erzeugt werden. In manchen, allerdings sehr seltenen Fällen kann es aber zu überraschenden Ergebnissen führen.

Betrachten wir die folgende Prozedur:

```
PR FOO  :X
 DRUCKE WERT  :X
ENDE
```

Hier ist X ein Variablenname, dessen Wert ein weiterer Variablenname ist, dessen Wert dann ausgedruckt werden soll. Schauen wir uns einige Beispiele an.

```
SETZE "A 3
FOO "A
3
```

```
SETZE "B 5
FOO "B
5
```

Dies funktioniert wie erwartet. Probieren wir

```
SETZE "X 6
FOO "X
X
```

Wie ist das zu erklären? Untersuchen wir zunächst das Beispiel FOO "A.

Privatbibliothek von FOO

Variablenname	Wert
X	A

Die Variable A ist nicht als Name in der Privatbibliothek von FOO enthalten. Deshalb führt der Befehl DRUCKE WERT :X, also DRUCKE WERT "A dazu, daß der Variablenname A in der Bibliothek der globalen Variablen gesucht wird. Logo findet ihn dort und weist ihm den Wert 3 zu, der dann ausgedruckt wird. Entsprechendes gilt für den Aufruf FOO "B.

Beim Aufruf FOO "X sieht die Privatbibliothek von FOO folgendermaßen aus:

Privatbibliothek von FOO

Variablenname	Wert
X	X

In diesem Beispiel ist X selbst der Wert von X. Bei Ausführung des Befehls DRUCKE WERT :X wird zunächst :X durch X ersetzt. Es ist also der Wert der Variablen X auszudrucken. Da X in der Privatbibliothek von FOO vorkommt, wird nun (im Gegensatz zum Fall FOO "A) der Wert von X aus *dieser* Bibliothek genommen. Der Wert der Variablen X ist in dieser Bibliothek aber X und dieser Wert wird ausgedruckt.

Natürlich ist das obige Beispiel ziemlich "verdreht" konstruiert, um dieses Phänomen überhaupt demonstrieren zu können. Wenn wir nur die Werte von gewissen Variablen ausdrucken wollten, so hätten wir dies mit Hilfe der Funktion

```
PR BAR  :X
 DRUCKE  :X
ENDE
```

einfacher haben können.

6.2.2 Fallstudien zur dynamischen und statischen Variablenbindung

Lassen Sie uns noch einmal zum MEHRWERTSTEUER / BRUTTOPREIS - Problem zurückkehren. Wir werden noch zwei weitere Versionen dieses Funktionenpaares betrachten und dabei jede Variante durch eine kleine "Demonstrations"-Prozedur ergänzen. Für die beiden schon aus Abschnitt 6.2.1 bekannten Versionen sehen die Demonstrations-Prozeduren folgendermaßen aus:

```
PR DEMO
 DRUCKEZEILE  BRUTTOPREIS  200  14
ENDE

PR DEMO.V2
 LOKAL  "MEHRWERTSTEUERSATZ
 SETZE  "MEHRWERTSTEUERSATZ 14
 DRUCKEZEILE  BRUTTOPREIS.V2 200
ENDE
```

Sowohl der Aufruf DEMO als auch der Aufruf DEMO.V2 führt wie erwartet zum Ausdruck des Wertes 228.00.

Nehmen wir einmal an, wir hätten in einem Anfall von Leichtsinnigkeit in den kontextgebundenen V2-Versionen sowohl die in MEHRWERTSTEUER.V2 vorkommende freie Variable MEHRWERTSTEUERSATZ als auch die in BRUTTOPREIS.V2 vorkommende lokale Variable NETTOPREIS jeweils mit dem Namen X bezeichnet. Aus den V2-Versionen entstünden dann die folgenden ebenfalls kontextabhängigen dritten Versionen dieser Funktionen:

```
PR MEHRWERTSTEUER.V3  :NETTOPREIS
 RUECKGABE  :NETTOPREIS  *  :X / 100
ENDE

PR BRUTTOPREIS.V3  :X
 RUECKGABE  :X + MEHRWERTSTEUER.V3  :X
ENDE
```

Nehmen wir weiterhin an, diese Funktionen würden durch den Aufruf DEMO.V3 der folgenden Demo-Version aktiviert:

```
PR DEMO.V3
 LOKAL  "X
 SETZE  "X  14
 DRUCKEZEILE  BRUTTOPREIS.V3  200
ENDE
```

Dann erhielten wir als Ausdruck an Stelle des Wertes 228.00 den Wert 600.00. Dieses Ergebnis ergibt sich folgendermaßen aus dem Verfahren der dynamischen Bindung: X ist sowohl eine lokale Variable in DEMO.V3 als auch in BRUTTOPREIS.V3; zudem kommt X als freie Variable in der Funktion MEHRWERTSTEUER.V3 vor. In DEMO.V3 und in MEHRWERTSTEUER.V3 steht X für den Mehrwertsteuersatz; in BRUTTOPREIS.V3 dagegen für den Nettobetrag.

Lassen Sie uns jetzt die Privatbibliotheken der drei Prozeduren beobachten. Vor dem Aufruf von DEMO.V3 sind alle drei Bibliotheken leer. Nach dem Aufruf von DEMO.V3 wird X als lokale Variable dieser Prozedur erzeugt und erhält den Wert 14 zugeordnet. Durch den Aufruf BRUTTOPREIS.V3 200 wird X als lokale Variable von BRUTTOPREIS.V3 erzeugt und erhält den Wert 200. Von BRUTTOPREIS.V3 wird schließlich auch die Funktion MEHRWERTSTEUER.V3 aufgerufen. Der Wert der freien Variablen X kann nicht in der Privatbibliothek von MEHRWERTSTEUER.V3 ermittelt werden. Deshalb schaut Logo in der Privatbibliothek der aufrufen-

den Prozedur BRUTTOPREIS.V3 nach, findet die Variable X dort, weist ihr den Wert 200 zu und arbeitet mit diesem Wert in MEHRWERTSTEUER.V3 weiter: die Mehrwertsteuer zum Nettobetrag von 200 DM beim Mehrwertsteuersatz von 200 % wird korrekt als 400.00 DM und der Bruttopreis schließlich als 600.00 DM ermittelt. Der "richtige" Wert von X zum Zeitpunkt des Aufrufs von MEHRWERTSTEUER.V3, der in der Privatbibliothek von DEMO.V3 zu finden gewesen wäre, war sozusagen durch den Aufruf von BRUTTOPREIS.V3 "verdeckt" worden. Das folgende Schema soll das Verständnis für die "Bibliotheksführung" im zeitlich-dynamischen Verlauf unterstützen.

Privatbibliotheken:

DEMO.V3		BRUTTOPREIS.V3		MEHRWERTSTEUER.V3	
Name	Wert	Name	Wert	Name	Wert
vor dem Aufruf von DEMO.V3:					
leer	leer	leer	leer	leer	leer
nach dem Aufruf von DEMO.V3, aber vor dem Aufruf von BRUTTOPREIS.V3:					
X	14	leer	leer	leer	leer
nach dem Aufruf von BRUTTOPREIS.V3, aber vor dem Aufruf von MEHRWERTSTEUER.V3:					
X	14	X	200	leer	leer
nach dem Aufruf von MEHRWERTSTEUER.V3:					
X	14	X	200	NETTOPREIS	200

Das Problem ist allerdings nur dadurch entstanden, daß wir nicht durchgängig mit lokalen Variablen sondern partiell auch mit freien Variablen gearbeitet und daß wir, die Namen der Variablen außerordentlich ungeschickt gewählt haben.

Wer bei der Namensvergabe "freie Hand" haben will, darf deshalb nicht mit freien Variablen arbeiten. Die folgende vierte Version zeigt schließlich, daß diese Prozeduren, in denen dieselben Namen wie in der dritten Version gewählt wurden, das "richtige" Ergebnis produzieren, wenn man ausschließlich mit lokalen Variablen arbeitet (wozu man natürlich alle Variablen in der Parameterliste mitführen muß).

```
PR MEHRWERTSTEUER.V4  :NETTOPREIS  :X
 RUECKGABE  :NETTOPREIS  *  :X / 100
ENDE
```

```
PR BRUTTOPREIS.V4  :X  :MEHRWERTSTEUERSATZ
 RUECKGABE  :X + MEHRWERTSTEUER.V4  :X  :MEHRWERTSTEUERSATZ
ENDE
```

Die zugehörige Demo-Version

```
PR DEMO.V4
 DRUCKEZEILE  BRUTTOPREIS.V4  200  14
ENDE
```

führt jetzt wieder zum erwarteten Ergebnis 228.00 DM.

Eine zweite, weit verbreitete Methode der Variablenbindung ist die sogenannte **statische Variablenbindung** (englisch: static binding bzw. static scoping). Bei dieser Methode ergibt sich der Wert einer freien Variablen grob gesprochen aus der jeweiligen "Definitionsumgebung", also aus dem Programmtext, in den der Aufruf der freien Variablen eingebettet ist. Das dynamische Laufzeitverhalten spielt dagegen keine Rolle. Im angelsächsischen Sprachraum bezeichnet man die Methode der statischen Bindung deshalb auch als lexical binding bzw. lexical scoping. Sie wird meist bei compilierten Sprachen wie zum Beispiel Pascal angewandt. Zum Vergleich wollen wir uns abschließend noch die entsprechenden Pascal-Versionen des MEHRWERTSTEUER / BRUTTOPREIS - Funktionenpaares anschauen. (Dies aus Platzgründen allerdings nur für die kritischen kontextabhängigen Versionen 2 und 3). An dieser Stelle kann natürlich nicht die Pascal-Syntax entwickelt werden. Aber diese Beispiele sind ja erstens wegen ihrer Einfachheit praktisch selbsterklärend und zweitens völlig parallel zu den entsprechenden Logo-Prozeduren.

```
Program demo_statische_bindung;

procedure demo_v2;
 var mehrwertsteuersatz: real;
 function mehrwertsteuer_v2(nettopreis: real): real;
  begin
    mehrwertsteuer_v2 := nettopreis * mehrwertsteuersatz / 100
  end;
 function bruttopreis_v2 (nettopreis: real): real;
  begin
    bruttopreis_v2 := nettopreis + mehrwertsteuer_v2(nettopreis)
  end;
begin (* demo_v2 *)
  mehrwertsteuersatz := 14;
  writeln(bruttopreis_v2(200) : 8 : 2 );
end;

procedure demo_v3;
  var x: real;
  function mehrwertsteuer_v3(nettopreis: real): real;
    begin
      mehrwertsteuer_v3 := nettopreis * x / 100
    end;
```

```
   function bruttopreis_v3(x: real): real;
     begin
       bruttopreis_v3 := x + mehrwertsteuer_v3(x)
     end;
 begin (* demo_v3 *)
   x := 14;
   writeln(bruttopreis_v3(200) : 8 : 2 );
 end;

 begin (* demo_statische_bindung*)
   demo_v2;
   demo_v3;
 end.
```

Das Kommando R (für RUN) zur Programmausführung führt zum Ausdruck der folgenden beiden Zeilen:

228.00
228.00

Die Methode der statischen Bindung hat also auch in der dritten Version zum erwarteten Ergebnis geführt. Beim Aufruf von mehrwertsteuer__v3 wurde die Variable x mit dem Wert belegt, der sich aus dem (lexikalischen) Programmtext und nicht aus dem (dynamischen) Aufrufverhalten der Prozeduren ergibt.

Man könnte sich nun fragen, warum nicht auch Lisp bzw. Logo das Verfahren der statischen Bindung praktizieren, das hier im Vergleich zu dem "besseren" Ergebnis geführt hat. Die Bewertung der einzelnen Bindungs-Methoden hängt jedoch auch noch von anderen Gesichtspunkten ab. Das Variablenkonzept von Lisp bzw. Logo ermöglicht zum Beispiel Dinge, wie die Iterierung der Wertrückgabe (Variable als Werte von Variablen) oder die Erzeugung neuer Variabler zur Laufzeit von Programmen, die in Pascal nicht möglich sind.

Die Probleme, welche sich aus den verschiedenen Bindungs-Methoden ergeben, sind prinzipieller Natur und lassen sich nicht grundsätzlich vermeiden. (Es scheint nicht *die* ideale Methode der Variablenbindung zu geben). In der Arbeit "The Art of the Interpreter" (MIT AI Memo No. 453) haben G.L.Steele und G.J.Sussman diesen Problemkreis analysiert.

Obwohl in den meisten Lisp-Versionen die Technik der dynamischen Variablenbindung praktiziert wird, gibt es gewisse Versionen, so zum Beispiel die Sprache SCHEME, welche andere Verfahren der Variablenbindung anwenden. SCHEME verwendet die Methode der statischen Bindung. In anderen Lisp-Versionen werden die Probleme, die sich aus der dynamischen Variablenbindung ergeben, durch das sogenannte "closure"-Konzept vermieden, das in den mir bekannten Logo-Versionen jedoch noch nicht realisiert ist (zum closure-Konzept siehe z.B.: R.Wilensky: "LISPcraft"). Dies hängt sicher nicht zuletzt damit zusammen, daß die meisten Logo-Versionen für relativ kleine "personal computer" entwickelt wurden.

6.3 Eine Standardmethode zur Einführung von lokalen Variablen

Eine sehr natürliche Methode, lokale Variable einzuführen, besteht darin, daß man eine Prozedur aufruft, in der diese Variablen als Eingabeparameter vorkommen. Mit dieser Methode kann man auch in solchen Logo-Versionen lokale Variable erzeugen, die nicht über das Sprachelement LOKAL (beziehungsweise LOCAL) verfügen. Diese Vorgehensweise wird am besten durch ein Beispiel erklärt.

Wir greifen dazu nochmals das Problem des Tilgungsplanes auf (siehe Abschnitt 6.1). Die Variablen RESTSCHULD, LAUFZEIT und der Zinsfaktor Q spielten dabei die Rolle von lokalen Variablen. Für den Aufruf der Prozedur TILGUNGSPLAN sind sie irrelevant; auch nach dem Ablauf der Prozedur spielen sie keine Rolle mehr. Nur solange die Prozedur TILGUNGSPLAN läuft, sind sie von Bedeutung.

Die folgende Prozedur TILGUNGSPLAN dient nur dazu, die Hilfsprozedur TILGUNGSPLAN.1, das eigentliche "Arbeitspferd", aufzurufen. Während die Eingabeparameter von TILGUNGSPLAN genau dem gegebenen Problem entsprechen, verfügt TILGUNGSPLAN.1 über weitere Eingabeparameter, die die Rolle der lokalen Variablen übernehmen. (Zinssatz wie üblich in Hundertsteln).

```
PR TILGUNGSPLAN  :ANFANGSDARLEHEN  :ZINSSATZ  :ANNUITAET
 TILGUNGSPLAN.1  :ANFANGSDARLEHEN  (1+:ZINSSATZ/100)  :ANNUITAET 0
ENDE

PR TILGUNGSPLAN.1  :RESTSCHULD  :Q  :ANNUITAET  :LAUFZEIT
 WENN  NICHT  :RESTSCHULD  > 0  DANN  RUECKKEHR
 DRUCKE  :LAUFZEIT
 DRUCKE  LEERWORT
 DRUCKEZEILE  :RESTSCHULD
 TILGUNGSPLAN.1  ( :RESTSCHULD * :Q  -  :ANNUITAET )  :Q  :ANNUITAET ___
 ______________  ( :LAUFZEIT  + 1 )
ENDE
```

Beim Aufruf von TILGUNGSPLAN.1 durch TILGUNGSPLAN werden die Eingabeparameter auf die richtigen Anfangswerte gesetzt: die Restschuld ist zu Beginn natürlich gleich dem Anfangsdarlehen, der Wert des Zinsfaktors Q ist gleich (1 + :ZINSSATZ / 100) und die Laufzeit ist am Anfang 0.

Die obige Hilfsprozedur TILGUNGSPLAN.1 ist völlig autonom. An Stelle von

TILGUNGSPLAN 67000 4.5 6000

kann man auch direkt

TILGUNGSPLAN.1 67000 1.045 6000 0

aufrufen.

Aufgabe 6.6: Schreiben Sie eine kontextabhängige Prozedur TILGUNGSPLAN.1 :RESTSCHULD :LAUFZEIT , welche die Werte der Variablen :ZINSSATZ und :ANNUITAET von TILGUNGSPLAN übernimmt und nur von dieser Prozedur aus aufgerufen werden darf.

Die Prozeduren TILGUNGSPLAN und TILGUNGSPLAN.1 können als Beispiele eines allgemeinen Schemas zur Erzeugung von lokalen Variablen angesehen werden. Nehmen wir an, wir wollen in der Prozedur FOO die lokale Variable L erzeugen. Dann können wir stets nach der folgenden Methode verfahren.

```
PR FOO   Parameter von FOO
 Initialisierungsroutinen von FOO
 Aufruf von FOO.1 mit geeigneten Werten fuer die Parameter von FOO.1
        (insbesondere einem geeigneten Anfangswert fuer die neue,
        lokale Variable L )
ENDE

PR FOO.1  Parameter von FOO.1 (darunter L )
 Prozedurkoerper von FOO.1
ENDE
```

Als ein weiteres Beispiel in diesem Zusammenhang wollen wir die (näherungsweise) Berechnung der Exponentialfunktion betrachten. Im Hinblick auf die mathematische Begründung des Verfahrens, die an dieser Stelle zu sehr vom eigentlichen Thema ablenken würde, sei auf das 1977 im Klett Verlag (Stuttgart) erschienene Buch "Elementarmathematik vom algorithmischen Standpunkt" (besonders Abschnitt 3.5 ff) von A.Engel verwiesen.

```
PR EXP  :X
 RUECKGABE  EXP.1  :X  1
ENDE

PR EXP.1  :X  :H
 SETZE  "H   ( :X / ( 3*3*3*3*3*3*3*3*3*3 ) )
 WENN  :X < 0  DANN SETZE  "H  (-1) * :H
 WIEDERHOLE 10  [ SETZE  "H  ( :H * ( 3 + 4 * :H * :H ) ) ]
 SETZE  "H  ( :H + QW ( 1 + :H * :H ) )
 WENN  :X < 0  DANN RUECKGABE  1 / :H
 RUECKGABE  :H
ENDE
```

Die lokale Variable H in EXP.1 spielt die Rolle einer Hilfsvariablen. Ihre Bestückung mit dem Wert 1 beim Aufruf von EXP.1 in EXP ist willkürlich, denn dieser "Anfangswert" wird in EXP.1 sofort überschrieben.

Einige Beispiele:

EXP 1
ERGEBNIS: 2.71828

EXP 5.5
ERGEBNIS: 244.69

EXP (-4)
ERGEBNIS: 0.0183158

EXP 10
ERGEBNIS: 22026.1

Aufgabe 6.7: (a) Informieren Sie sich in dem obengenannten Buch von A.Engel über die mathematische Begründung des in EXP.1 angewandten Verfahrens.
(b) Vergleichen Sie die obigen Werte mit denen in einer geeignetenTabelle. Stellen Sie weitere Vergleiche an.

6.4 Tail Recursion / Endrekursion

Die Prozedur

```
PR ARITHMETISCHE.FOLGE  :A  :D  :MAX
 WENN  :A  >  :MAX  DANN RUECKKEHR
 DRUCKEZEILE  :A
 ARITHMETISCHE.FOLGE  ( :A  +  :D )  :D  :MAX
ENDE
```

ist zweifellos rekursiv. Wenn wir sie aufrufen, zum Beispiel durch ARITHMETISCHE.FOLGE 1 2 100000, so läuft sie im Gegensatz zu den in Kapitel 5 betrachteten rekursiven Funktionen sehr lange, ohne daß der Speicher gesprengt wird. Ein Beispiel:

```
ARITHMETISCHE.FOLGE 1 2 1000
1
3
5
7
9
11
...
999
```

Aufgabe 6.8: Schreiben Sie eine Prozedur GEOMETRISCHE.FOLGE.

Aufgabe 6.9: Der Erfinder des Schachspiels durfte als Belohnung einen Wunsch äußern. Er wünschte sich einige Reiskörner. Und zwar ein Korn auf dem ersten Feld des Schachbretts, zwei auf dem zweiten, vier auf dem dritten, acht auf dem vierten, u.s.w.: auf jedem Feld die doppelte Anzahl der Körner des vorherigen Feldes.

(a) Wieviele Reiskörner müßten auf dem 64-ten Feld angehäuft werden?
(b) Wieviele Reiskörner wären dann insgesamt auf dem Schachbrett?

Zurück zur Prozedur ARITHMETISCHE.FOLGE. Sie ist ein Beispiel für eine spezielle Form der Rekursion, die in englisch als **tail recursion** oder last line recursion bezeichnet wird. In der deutschen Sprache gibt es meines Wissens keine genormte Bezeichnung; im folgenden wird diese Form der Rekursion als **Endrekursion** bezeichnet.

Allgemein gesprochen sind endrekursive Prozeduren solche Prozeduren, in denen der rekursive Aufruf nur an einer einzigen Stelle und zwar als letzter Befehl der Prozedur auftritt. Dem endrekursiven Aufruf darf auch kein RUECKGABE-Befehl vorangehen, da sonst der RUECKGABE-Befehl der letzte Befehl wäre. Die Fakultätsfunktion aus Kapitel 5 (Abschnitt 5.6.3.1) ist also nicht endrekursiv.

Im Falle einer *echten Rekursion*, wie zum Beispiel von FAK oder FIB in Abschnitt 5.6.3.1, muß sich Logo merken, an welche Stelle es die im Rücklauf der Rekursion ermittelten (Zwischen-) Werte jeweils zur Weiterverarbeitung zurückgeben muß. Zu diesem Zweck baut Logo im Speicher des Computers einen sogenannten Rekursionsstack auf. Wenn dieser Rekursionsstack wegen der Tiefe der Rekursion zu groß wird, wird der Speicher gesprengt. Logo bringt dann die Meldung "SPEICHER VOLL !".

Das Ablaufdiagramm zum obigen Aufruf zeigt, daß endrekursive Prozeduren einen "leeren Rücklauf" haben. Nach dem rekursiven Aufruf werden in der Prozedur weder Funktionswerte weitergereicht, noch wird etwas ausgedruckt. Das einzige, was im Rücklauf passiert ist, daß die Kontrolle an die jeweils aufrufende Stelle zurückgegeben wird. Um zum Ausgangspunkt vor dem ersten Aufruf zurückzukehren, ist es eigentlich überflüssig, daß sich Logo wie im Falle der echten Rekursion alle Zwischenstationen merkt. Es braucht sich nur die Stelle zu merken, von der aus der endrekursive Aufruf erfolgte, um nach dem Durchlaufen der Aufrufkette wieder zu dieser Stelle zurückzukehren.

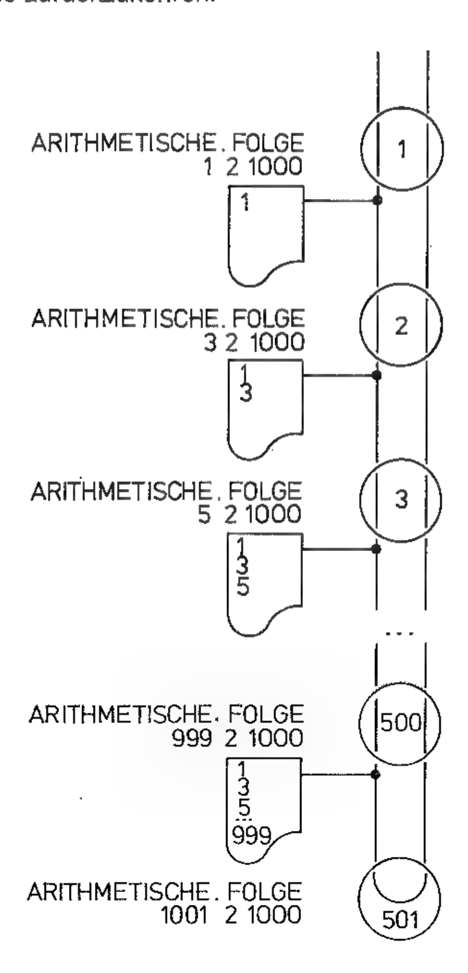


Abbildung 6.1:
Aufrufkette zur arithmetischen Folge

Logo erkennt endrekursive Prozeduren automatisch und führt sie nach einem optimierenden Interpretationsschema aus, ohne daß ein Rekursionsstack aufgebaut wird. Derartige Prozeduren können also im Prinzip beliebig lange laufen.

Besonders beliebt sind endrekursive Prozeduren in der Igel-Geometrie. Auch dazu noch ein Beispiel:

```
PR SPIRALE :SEITE :WINKEL :ZUWACHS
 VORWAERTS :SEITE
 RECHTS :WINKEL
 SPIRALE ( :SEITE + :ZUWACHS ) :WINKEL :ZUWACHS
ENDE
```

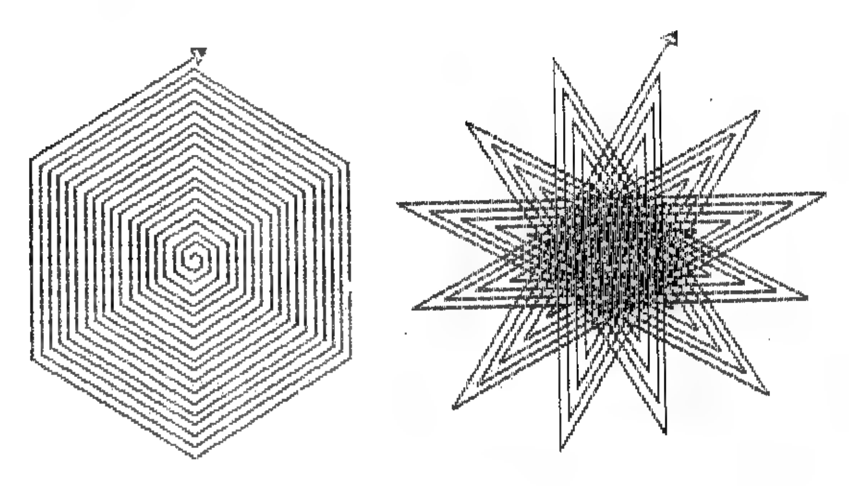


Abbildung 6.2: SPIRALE 2 60 1 und SPIRALE 5 150 3

6.5 Eine Methode zur Ersetzung gewisser rekursiver Funktionen durch endrekursive Prozeduren

Da endrekursive Prozeduren nicht dazu führen, daß durch den Aufbau eines Rekursionsstack der Speicher gesprengt wird, liegt es nahe, daß man versuchen wird, rekursive Prozeduren durch endrekursive zu ersetzen. Die folgende sehr einfache Methode ist nicht auf alle rekursiven Prozeduren anwendbar, aber in bestimmten Fällen führt sie zum Erfolg.

Wir wollen sie am Beispiel der Funktion ELEMENTEZAHL aus Kapitel 5 erläutern. Zunächst die alte Version:

```
PR ELEMEMTEZAHL :L
 WENN :L = [ ] DANN RUECKGABE 0
 RUECKGABE 1 + ELEMENTEZAHL OHNEERSTES :L
ENDE
```

Die folgende Prozedur dient nur dazu, eine "große" Liste herzustellen, die die Zahlen von 1 bis 300 als Elemente enthält.

```
PR GROSSE.LISTE.MACHEN
 SETZE  "GL  [ ]
 LOKAL  "I
 SETZE  "I 0
 WIEDERHOLE 300 [ SETZE  "I  :I+1   SETZE  "GL  MITLETZTEM  :I  :GL ]
ENDE
```

GROSSE.LISTE.MACHEN
ELEMENTEZAHL :GL
SPEICHER VOLL!

Nun die neue Version:

```
PR ELEMENTEZAHL  :L
 LOKAL  "I
 SETZE  "I 0
 ELEMENTEZAHL.1  :L
 RUECKGABE  :I
ENDE

PR ELEMENTEZAHL.1  :L
 WENN  :L = [ ]  DANN RUECKKEHR
 SETZE  "I  :I+1
 ELEMENTEZAHL.1  ( OHNEERSTES  :L )
ENDE
```

Die in der aufrufenden Prozedur ELEMENTEZAHL erzeugte lokale Variable I ist auch in der aufgerufenen Prozedur ELEMENTEZAHL.1 bekannt. Die als Funktionswert auszugebende Elementezahl wird in der Hilfsprozedur ELEMENTEZAHL.1 als Wert von I aufgebaut. Für die Variable I gelten die in Abschnitt 6.2 beschriebenen Bindungsregeln. Die Hilfsprozedur ELEMENTEZAHL.1 ist endrekursiv und kann deshalb beliebig lange laufen. (Man kann die Rolle der Hilfsvariablen I und der Hilfsprozedur ELEMENTEZAHL.1 auch folgendermaßen deuten: die Variable I zählt im wesentlichen, wie oft die Prozedur ELEMENTEZAHL.1 aufgerufen wurde).

Ein Beispiel:

ELEMENTEZAHL :GL
ERGEBNIS: 300

Aufgabe 6.10: Schreiben Sie neue endrekursive Versionen für die folgenden Funktionen aus Abschnitt 5.6.3: FAK, PICKE, ELEMENT?, ERSETZE, SPIEGELUNG, BUCHSTABENLISTE

Aufgabe 6.11: Warum funktioniert das Verfahren nicht bei FIB ?

6.6 Prozeduren und Funktionen, deren Argumente selbst Funktionen sind (Funktionen von Funktionen)

6.6.1 Wertetafeln

Die Wertetafel einer Funktion ist eine Tabelle, in der neben jedem Argumentwert der Variablen der entsprechende Funktionswert notiert ist. Als Namen für die Variablen verwendet man meist x, gelegentlich auch y und z; im Grunde genommen kann der Name aber völlig beliebig gewählt werden. Hier als Beispiel ein winziger Ausschnitt aus der Wertetafel der Quadratfunktion f(x) = x * x :

Argument	Funktionswert
x	f(x)
...	...
2	4
3	9
4	16
...	...

Bei der Untersuchung von technisch / naturwissenschaftlichen oder wirtschaftlichen Fragestellungen benötigt man sehr oft die Wertetafeln bestimmter Funktionen. Wir wollen deshalb eine Prozedur WERTETAFEL schreiben, die eben diese Aufgabe übernimmt. Was aber sind die "natürlichen" Parameter einer Prozedur WERTETAFEL ? Sicherlich der Anfangswert, der Endwert und die Schrittweite. Irgendwie muß auch die Funktion, von der wir die Wertetafel erstellen wollen, in diese Prozedur eingehen. Damit wir nicht für jede neue Funktion eine andere Prozedur WERTETAFEL schreiben müssen, übergeben wir auch die Funktion, von der die Wertetafel zu erstellen ist, als Eingabeparameter an die Prozedur WERTETAFEL.

```
PR WERTETAFEL  :FUNKTION  :ANFANG  :ENDE  :SCHRITT
 WENN  :ANFANG  >   :ENDE   DANN  RUECKKEHR
 LOKAL  "X
 SETZE  "X  :ANFANG
 ( DRUCKEZEILE  :X  LEERWORT  TUE  :FUNKTION )
 WERTETAFEL  :FUNKTION  ( :ANFANG +  :SCHRITT )  :ENDE  :SCHRITT
ENDE
```

LEERWORT ist die schon des öfteren benutzte Funktion aus Kapitel 4, Abschnitt 4.2.4. Der TUE-Befehl wertet Listen aus. Wir haben ihn bereits in Kapitel 5, Abschnitt 5.7, kennengelernt. Zur Wiederholung noch ein Beispiel:

```
TUE [ QW 9 ]
ERGEBNIS: 3.0
```

Im Programm WERTETAFEL durchläuft die Variable X eine bestimmte Menge von Argumentwerten. Sie muß dazu explizit an das jeweilige Argument gebunden werden. Dies geschieht durch den SETZE-Befehl. Hier ein Beispiel:

```
SETZE "X 6.25
TUE [ QW :X ]
ERGEBNIS: 2.5
```

(Vergessen Sie nicht, die soeben erzeugte globale Variable X wieder zu löschen).

Die Funktion, von der wir die Wertetafel wollen, muß also in Listenform eingegeben werden. Im folgenden sind einige vollständige Aufrufbeispiele der Prozedur WERTETAFEL gegeben. (Die Formatierung der Ausgabe ist vom Computer abhängig. Im allgemeinen wird das obige Programm keinen stellengerechten Ausdruck erzeugen; siehe: allgemeine Hinweise am Anfang dieses Buches. Ein formatierter Ausdruck ist aber durch die Verwendung der Hilfsprozedur DRUCKE.STELLENGERECHT aus Kapitel 9, Abschnitt 9.3, erreichbar).

```
WERTETAFEL [ 2 * SIN (:X*:X) ] (-1) 3 0.5

-1.0    0.0349048
-0.5    0.0087262
 0.0    0.0
 0.5    0.0087262
 1.0    0.0349048
 1.5    0.0785172
 2.0    0.139513
 2.5    0.217727
 3.0    0.312869
```

```
WERTETAFEL [ 3 * :X + 10 ] 0 20 4

 0    10
 4    22
 8    34
12    46
16    58
20    70
```

Einen kleinen Schönheitsfehler hat die Prozedur WERTETAFEL noch: die "Laufvariable" muß in dieser Version immer den Namen X haben. In der folgenden Version von WERTETAFEL ist der *Name* der Laufvariablen ein Eingabeparameter und somit selbst variabel.

```
PR WERTETAFEL  :FUNKTION  :A  :E  :S  :VARIABLENNAME
 WENN  :A   >  :E   DANN  RUECKKEHR
 LOKAL   :VARIABLENNAME
 SETZE   :VARIABLENNAME  :A
 ( DRUCKEZEILE  WERT :VARIABLENNAME  LEERWORT  TUE  :FUNKTION )
 WERTETAFEL  :FUNKTION  ( :A + :S )  :E  :S  :VARIABLENNAME
ENDE
```

Beim folgenden Aufruf ist zum Beispiel :VARIABLENNAME = Y und WERT :VARIABLENNAME (= :Y) eine Zahl zwischen 1 und 5.

WERTETAFEL [3 * :Y * :Y + 5 * :Y - 16] 1 5 0.5 "Y

```
1.0    -8.0
1.5    -1.75
2.0     6.0
2.5    15.25
3.0    26.0
3.5    38.25
4.0    52.0
4.5    67.25
5.0    84.0
```

Einige weitere Beispiele:

WERTETAFEL [(SIN :Z) * (COS :Z)] 0 10 2 "Z

```
 0.0    0.0
 2.0    0.0348782
 4.0    0.0695865
 6.0    0.103956
 8.0    0.137819
10.0    0.17101
```

WERTETAFEL [(SIN :W)*(SIN :W) + (COS :W)*(COS :W)] 0 360 1 "W

```
  1.0    1.0
  2.0    1.0
  3.0    1.0
  ...
360.0    1.0
```

Aufgabe 6.12: In den bisherigen Realisierungen der Prozedur WERTETAFEL war FUNKTION eine Variable, welche (verpackt in einer Liste) die auszuwertende Funktion selbst enthielt. Für jemanden, der viel mit selbstdefinierten Funktionen arbeitet, kann es gelegentlich nützlich sein, wenn nicht die Funktion selbst sondern nur ihr Name als Parameter übergeben werden muß. Wenn wir dieser Form der Prozedur den Namen WERTETAFEL.FN (FN für Funktionsname) geben, so müßte ein Aufruf zum Beispiel folgendermaßen lauten:

WERTETAFEL.FN "EXP (-5) 20 0.01 "X

Dabei sei EXP die im Abschnitt 6.3 behandelte Exponentialfunktion. Schreiben Sie eine Wertetafel-Prozedur, welche dies leistet.

Aufgabe 6.13: Schreiben Sie eine Prozedur
SUPER.WERTETAFEL :FKT :A :E :S :VARIABLENNAME ,
die ihrerseits entweder WERTETAFEL oder WERTETAFEL.FN aufruft, je nachdem, ob der Wert des Parameters FKT die Funktion selbst oder der Name der Funktion ist.

Aufgabe 6.14: (Ein kleines Projekt)
Schreiben Sie entsprechende Prozeduren FUNKTIONSSCHAUBILD, FUNKTIONSSCHAUBILD.FN und SUPER.FUNKTIONSSCHAUBILD, mit denen Sie die Funktionswerte graphisch auf dem Bildschirm darstellen können.

6.6.2 Der Differenzenquotient

Ein weiteres typisches Beispiel für eine Funktion, deren Argumente wieder Funktionen sind, ist der Differenzenquotient. Er läßt sich als die Steigung der Sekante s deuten, die zwei Punkte eines Funktionsgraphen miteinander verbindet. Die Situation wird am besten durch eine Skizze verdeutlicht:

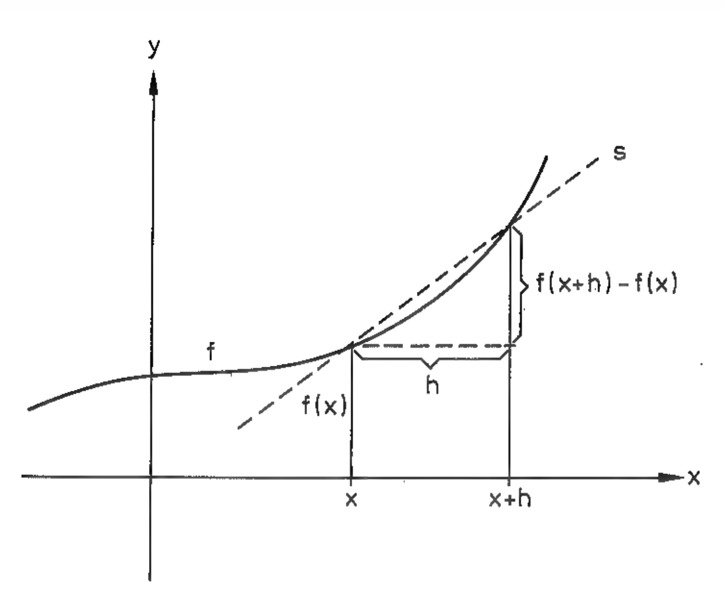


Abbildung 6.3: Der Differenzenquotient

Der **Differenzenquotient** der Funktion f an der Stelle x bei der Schrittweite h ist definiert als

$$D(f,x,h) := (f(x+h) - f(x)) / h$$

(Wenn wir h gegen Null streben lassen, so geht der Differenzenquotient bei differenzierbaren Funktionen in den sogenannten **Differentialquotienten** über, der als die Tangentensteigung an der Stelle (x, f(x)) gedeutet werden kann).

In Logo können wir den Differenzenquotienten folgendermaßen definieren:

```
PR DIFFQUOT  :F  :X  :H
 RUECKGABE ( ( FA  :F (  :X  +  :H ) )  -  ( FA  :F  :X  ) ) / :H
ENDE
```

FA soll dabei FUNKTIONSAUSWERTUNG bedeuten.

```
PR FA  :F  :X
 RUECKGABE  TUE  :F
ENDE
```

Eine noch flexiblere Prozedur zur Funktionsauswertung werden wir im nächsten Abschnitt in Form der Funktion ANWENDUNG kennenlernen. Die obige Form

ermöglicht es jedoch, an Stelle eines Funktionsnamens einfach ihren "Funktionsterm" (wenn auch nur in der Variablen X) einzugeben. Auf das Thema "Funktionen als Rechenvorschriften" kommen wir in Abschnitt 6.6.5 zurück.

Einige Beispiele:

DIFFQUOT [5 * :X * :X * :X] 2.5 0.1
ERGEBNIS: 97.5494

DIFFQUOT [5 * :X * :X * :X] 2.5 0.01
ERGEBNIS: 94.1193

DIFFQUOT [5 * :X * :X * :X] 2.5 0.001
ERGEBNIS: 93.75

Probe (mit der "Ableitungsfunktion" 3 * 5 * :X * :X):
15 * 2.5 * 2.5
ERGEBNIS: 93.75

Hier noch einige weitere Beispiele mit den nachfolgenden trigonometrischen Funktionen SINUS und COSINUS:

```
PR SINUS  :X
 RUECKGABE  SIN  :X * 180 / PI
ENDE

PR COSINUS  :X
 RUECKGABE COS  :X  * 180 / PI
ENDE
```

Dabei sei PI die folgende konstante Funktion:

```
PR PI
 RUECKGABE  3.14159
ENDE
```

DIFFQUOT [SINUS :X] 2 0.5
ERGEBNIS: -0.621618

DIFFQUOT [SINUS :X] 2 0.2
ERGEBNIS: -0.503867

DIFFQUOT [SINUS :X] 2 0.1
ERGEBNIS: -0.460832

DIFFQUOT [SINUS :X] 2 0.01
ERGEBNIS: -0.419223

DIFFQUOT [SINUS :X] 2 0.001
ERGEBNIS: -0.414372

Probe (mit der Ableitung):
COSINUS 2
ERGEBNIS: -0.416133

Aufgabe 6.13: (a) Geben Sie den obigen Funktionen SINUS und COSINUS eine inhaltliche Deutung.
(b) Warum funktioniert der letzte Test nicht ohne weiteres mit den Logo-Grundfunktionen SIN bzw. COS an Stelle von SINUS und COSINUS ?
(c) Führen Sie zu den letzten sechs Ergebniswerten eine Monotoniebetrachtung (am besten anhand einer Skizze) durch und überprüfen Sie die Plausibilität dieser Werte kritisch.

Die folgenden Abschnitte 6.6.3 bis 6.6.5 wenden sich primär an den mathematisch interessierten Leser. In ihnen werden weitere vertiefte Anwendungen des Funktionskonzepts von Logo im mathematischen Kontext aufgezeigt.

6.6.3 Verkettung von Funktionen

Wohl die wichtigste Operation mit Funktionen ist ihre Verkettung. Sind f und g beliebige verkettbare Funktionen, so wird ihre Verkettung mit (f o g) bezeichnet (vergleiche Abbildung 2.4). Die Verkettung der Funktionen SIN und QW wird also zum Beispiel folgendermaßen geschrieben: (SIN o QW). Jede Funktion ist eindeutig durch das definiert, was sie mit ihren Argumenten macht:

(SIN o QW) (x) := SIN (QW (x)).

Also zum Beispiel (SIN o QW) (3) = SIN (QW (3)) = 0.030225.

An Stelle von der Verkettung spricht man in der Mathematik auch häufig vom **Produkt** zweier Funktionen. In Logo läßt sich die Funktionsverkettung folgendermaßen realisieren:

```
PR PRODUKT  :F  :G
 RUECKGABE ( SATZ  :F  :G )
ENDE
```

Von den Funktionen :F und :G sind dabei nur die Funktionsnamen einzugeben; zum Beispiel folgendermaßen:

PRODUKT "SIN "QW
ERGEBNIS: [SIN QW]

Da die Logo-Grundfunktion SATZ sowohl Wörter als auch Listen verarbeitet (siehe Kapitel 4, Abschnitt 4.3.1), hätte man dasselbe Ergebnis mit dem Aufruf PRODUKT [SIN] [QW] erreicht. Allgemein kann man die Funktionsnamen (ohne Argumente) in "gequoteter" Form oder auch als Listenelemente eingeben.

Im Gegensatz zur Funktion DIFFQUOT haben wir der Verkettung noch nicht die Argumente beigefügt, mit denen sie ausgewertet werden soll. Wir benötigen hierfür noch eine Funktion zur Erzwingung der Auswertung; sie sei ANWENDUNG genannt. ANWENDUNG hat zwei Parameter: eine Funktion und ein Argument. Sie wendet die in der Liste :FL enthaltene Funktion auf das Argument :X an.

```
PR ANWENDUNG  :FL  :X
 RUECKGABE  TUE  MITLETZTEM  :X  :FL
ENDE
```

An Stelle von TUE MITLETZTEM :X :FL hätte man auch TUE SATZ :FL :X schreiben können.

Einige Beispiele:

ANWENDUNG [QW] 9
ERGEBNIS: 3.0

ANWENDUNG PRODUKT "SIN "QW 3
ERGEBNIS: 0.030225

ANWENDUNG PRODUKT [INT] [QW] 5
ERGEBNIS: 2

In Direktausführungsmodus wird man (außer zu Testzwecken) die letzten Aufrufe in der einfacheren Form QW 9, SIN QW 3 oder INT QW 5 eingeben, ohne die PRODUKT-Funktion zu bemühen. Das Entscheidende an der PRODUKT-Funktion ist, daß man mit ihr Funktionen verknüpfen kann, die sich erst zur Laufzeit von Programmen (zum Beispiel aufgrund eines Eingabedialogs) ergeben.

Im folgenden soll an einem etwas umfangreicheren Beispiel eine Anwendungsmöglichkeit der Funktionsverkettung aufgezeigt werden. Dazu definieren wir uns zunächst einmal die folgenden sechs Funktionen:

```
PR F1  :Z
 RUECKGABE  :Z
ENDE

PR F2  :Z
 RUECKGABE  ( :Z - 1 ) / :Z
ENDE

PR F3  :Z
 RUECKGABE  1 / ( 1 - :Z )
ENDE

PR F4  :Z
 RUECKGABE  1 / :Z
ENDE

PR F5  :Z
 RUECKGABE  1 - :Z
ENDE

PR F6  :Z
 RUECKGABE  :Z / ( :Z - 1 )
ENDE
```

Falls der Wert der Variablen Z von 0 und 1 verschieden ist, sind alle diese Funktionen definiert, leicht auswertbar und beliebig miteinander verkettbar. Die erste Funktion F1 reicht ihr Argument unverändert weiter; man nennt sie auch die **identische Funktion**. Hier einige Beispiele:

ANWENDUNG PRODUKT "F2 "F4 2
ERGEBNIS: -1.0

F5 2
ERGEBNIS: -1

ANWENDUNG PRODUKT "F2 "F4 5
ERGEBNIS: -4.0

F5 5
ERGEBNIS: -4

Aufgabe 6.15: (a) Zeigen Sie für weitere Werte von Z, daß F5 und die Verkettung von F2 und F4 stets zu denselben Ergebniswerten führen.
(b) Begründen Sie allgemein, daß die Funktion F5 mit der Verkettung von F2 und F4 übereinstimmt.

Wir bezeichnen im folgenden die Funktion F5 auch als die Ersatzfunktion für die Verkettung von F2 und F4.

Wenn wir mit den oben gegebenen Funktionen F1, ..., F6 weiter experimentieren, dann erkennen wir, daß die Verkettung je zweier dieser Funktionen stets wieder zu einer Ersatzfunktion aus F1, ..., F6 führt.

Dieser Sachverhalt ist in prägnanter Form in der folgenden Tabelle dargestellt:

	F1	F2	F3	F4	F5	F6
F1	F1	F2	F3	F4	F5	F6
F2	F2	F3	F1	F5	F6	F4
F3	F3	F1	F2	F6	F4	F5
F4	F4	F6	F5	F1	F3	F2
F5	F5	F4	F6	F2	F1	F3
F6	F6	F5	F4	F3	F2	F1

Diese Tabelle ist folgendermaßen zu lesen: Die linke Spalte nennen wir die *Eingangsspalte*, die obere Zeile heißt *Eingangszeile*. Wenn wir eine bestimmte Funktionsverkettung betrachten, etwa die von F4 mit F3, so können wir deren Ersatzfunktion im Schnittpunkt der Zeile, in der F4 steht und der Spalte, in der F3 steht, ablesen; hier: F5. Es gilt also

(ANWENDUNG PRODUKT "F4 "F3 :X) = (F5 :X)

für jedes beliebige von Null und Eins verschiedene :X.

Beachten Sie, daß die Verkettung von Fi und Fj im allgemeinen zu einer anderen Ersatzfunktion führt als die Verkettung von von Fj und Fi (i und j seien dabei beliebige ganze Zahlen zwischen 1 und 6).

Aufgabe 6.16: (a) Erhärten Sie die Richtigkeit der oben gegebenen Tabelle durch weitere konkrete Auswertungsbeispiele.
(b) Bestätigen Sie die Richtigkeit der Tabelle allgemein durch algebraische Umformungen.

Den Teil der obigen Tafel, den man erhält, wenn man die Eingangsspalte und Eingangszeile wegläßt, wollen wir als das *Produktfeld* bezeichnen. Wir erkennen sofort: In jeder Zeile des Produktfeldes kommt jede der sechs Funktionen genau einmal vor.

Aufgabe 6.17: Suchen Sie weitere Gesetzmäßigkeiten in der obigen Tafel.

Nur der Vollständigkeit halber sei erwähnt, daß man Tafeln, wie die oben gegebene, auch als *Gruppentafeln* bezeichnet.

6.6.4 Potenzierung von Funktionen

Ein Produkt ist das Ergebnis einer Multiplikation. Die logische Weiterführung des Multiplizierens ist das Potenzieren. So kann man zum Beispiel die dritte Potenz von 2 als das Produkt der Multiplikation mit drei Faktoren 2 definieren.

Im Bereiche der Zahlen ist eine formale Zurückführung des Potenzierens auf das Multiplizieren in der folgenden Weise möglich (dabei sei n eine natürliche Zahl und pot(x,n) die n-te Potenz von x).

$$\begin{aligned} pot(x,n) &:= x && \text{(für } n = 1);\\ pot(x,n) &:= x * pot(x, n-1) && \text{(für } n > 1). \end{aligned}$$

Im vorigen Abschnitt haben wir gesehen, wie man auch für Funktionen eine Multiplikation einführen kann. Es liegt nun nahe, auch das Potenzieren von Funktionen zu definieren. Die folgende Funktion FPOT :F :N hat als Ergebnis die :N-te Potenz der Funktion :F. (Im Hinblick auf die Verwendung der Funktion FPOT im Direktausführungsmodus gelten dieselben Bemerkungen wie für die PRODUKT-Funktion).

```
PR FPOT  :F  :N
 WENN  :N = 1  DANN RUECKGABE  ( SATZ  :F )
 RUECKGABE  PRODUKT  :F   ( FPOT  :F  (:N - 1) )
ENDE
```

Aufgab 6.17: Harry Hacker behauptet, daß RUECKGABE (LISTE :F) an Stelle von RUECKGABE (SATZ :F) in FPOT genau dasselbe bewirkt. Setzen Sie sich mit dieser Behauptung auseinander.

FPOT legt also eine Liste mit :N Kopien der Funktion :F an. Zum Experimentieren definieren wir noch die Quadratfunktion

```
PR Q  :X
 RUECKGABE  :X * :X
ENDE
```

```
FPOT "Q 3
ERGEBNIS: [ Q Q Q ]

ANWENDUNG FPOT "Q 3 2
ERGEBNIS: 256
```

Logo hat den letzten Funktionsaufruf folgendermaßen ausgewertet (die runden Klammern sind hier nur zur Verdeutlichung gesetzt).

ANWENDUNG (FPOT "Q 3) 2 = Q Q Q 2 = Q Q 4 = Q 16 = 256

Weitere Beispiele:

ANWENDUNG FPOT [Q] 4 3
ERGEBNIS: 43046721

ANWENDUNG FPOT [QW] 4 64536
ERGEBNIS: 2.0

6.6.5 Anonyme Funktionen (Funktionen als Rechenvorschriften)

Rechenausdrücke, wie zum Beispiel 1 / x , geben automatisch Anlaß zur Betrachtung von *zugehörigen Funktionen*. Die zu 1 / x gehörende Funktion wird in der Mathematik meist folgendermaßen bezeichnet:

x —> 1 / x (∗).

Bei dieser Darstellungsform wird über eine Funktion gesprochen, ohne daß ihr ein bestimmter Name gegeben wird. Natürlich ist es durchaus möglich, jeder Funktion einen Namen zu geben. In Abschnitt 6.6.3 haben wir die durch (∗) gegebene Funktion zum Beispiel F4 genannt. Ein plausibler Name wäre auch INVERS. Man könnte sie als Logo-Funktion folgendermaßen realisieren:

```
PR INVERS  :X
 RUECKGABE 1 / :X
ENDE
```

INVERS 2
ERGEBNIS: 0.5

Es ist aber unter Umständen lästig und wirkt manchmal auch etwas schwerfällig, wenn man jeder dieser Funktionen unbedingt einen Namen geben muß. Deshalb gibt es in Lisp die Möglichkeit, Funktionen als Rechenausdrücke einzuführen. In Lisp wird dazu die sogenannte **Lambda**-Notation verwendet. Sie hat ihren Ursprung im Lambda-Kalkül, den der Logiker A.Church in der ersten Hälfte dieses Jahrhunderts entwickelte. Er konnte 1936 zeigen, daß die Klasse der rekursiven Funktionen mit der Klasse der durch den Lambda-Kalkül definierten Funktionen zusammenfällt. Eine grundlegende Diskussion der logisch-mathematischen Seite dieser Problematik ist in dem Buch "Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit" von Hans Hermes (Springer Verlag) zu finden.

In Logo gibt es zwar nicht, wie in Lisp, einen Lambda-Operator als Grundwort; wir können ihn aber leicht als neue Funktion definieren. Vorher wollen wir uns aber ansehen, wie der Lambda-Operator zu verwenden sein soll.

Bezeichnung durch Funktionsnamen	Beschreibung als Rechenausdruck	Bezeichnung in der Lambda/Notation
INVERS	x —> 1 / x	LAMBDA [[X] [1 / :X]]
Q	x —> x ∗ x	LAMBDA [[X] [:X ∗ :X]]
F6 (aus 6.6.3)	x —> x / (x-1)	LAMBDA [[X] [:X / (:X -1)]]

Die Funktion LAMBDA soll also als Eingabeparameter eine Liste haben, die aus zwei Teillisten besteht. Die erste Teilliste enthält den Namen der sogenannten "gebundenen" Variablen, die zweite Teilliste (im folgenden auch als LAMBDA-Körper bezeichnet) enthält die Rechenvorschrift.

Zur Auswertung des Lambda-Ausdruckes müssen wir nun noch, wie bei Funktionen üblich, einen konkreten Wert hinzufügen, an den die Variable (in den obigen Fällen X) zu binden ist. Genau wie beim Aufruf Q 5 soll der Aufruf

LAMBDA [[X] [:X * :X]] 5

zum Ergebnis 25 führen.

Es ist jetzt an der Zeit, die Funktion LAMBDA zu definieren.

```
PR LAMBDA  :L  :W
 LOKAL ERSTES ERSTES  :L
 SETZE ERSTES ERSTES  :L  :W
 RUECKGABE TUE LETZTES  :L
ENDE
```

Ein Beispiel:

LAMBDA [[X] [1 / :X]] 8
ERGEBNIS: 0.125

An dieser Stelle sei ausdrücklich darauf hingewiesen, daß die Variable X in der Liste der Eingabeparameter *nicht* mit einem Doppelpunkt zu versehen (und auch nicht zu quoten) ist; daß also nicht etwa

LAMBDA [[:X] [1 / :X]] 8 oder
LAMBDA [["X] [1 / :X]] 8

zu schreiben ist. Dies folgt aus der Art und Weise, wie Logo Listen analysiert (siehe Abschnitt 4.3.2.1). Der Logo-Editor ist hier etwas weniger konsequent, denn man kann die Eingabeparameter von Prozeduren mit Doppelpunkten versehen oder auch nicht:

```
PR QUADRAT.1  :X
 RUECKGABE  :X * :X
ENDE

PR QUADRAT.2  X
 RUECKGABE  :X * :X
ENDE
```

QUADRAT.1 3
ERGEBNIS: 9

QUADRAT.2 4
ERGEBNIS: 16

Weitere Beispiele zur LAMBDA-Funktion:

LAMBDA [[Y] [1 + :Y + :Y * :Y]] 2.4
ERGEBNIS: 9.16

LAMBDA [[Z] [:Z + 1 / QW :Z]]   64
ERGEBNIS: 64.125

Da die gebundene Variable bei der Auswertung des LAMBDA-Ausdrucks mit einem konkreten Wert belegt wird, ist es übrigens gleichgültig, mit welchem Namen wir sie belegen (natürlich vorausgesetzt, daß keine Namenskollision eintritt). Während also die Terme 1 / :X und 1 / :Y im allgemeinen verschieden sind, sind die Funktionen

LAMBDA [[X] [1 / :X]] und

LAMBDA [[Y] [1 / :Y]]

in dem Sinne gleich, daß sie, auf ein konkretes Argument angewandt, stets zu demselben Ergebnis führen.

Wie wir im letzten Beispiel (mit der Funktion QW) gesehen haben, können in den im LAMBDA-Körper auftretenden Rechenausdrücken auch wieder Funktionsnamen auftreten. Nichts spricht dagegen, daß diese Funktionen selbst wieder als LAMBDA-Ausdrücke geschrieben werden.

LAMBDA [[X] [2 * :X + 1 / LAMBDA [[X] [:X * :X]] :X]] 5
ERGEBNIS: 10.04

Man hätte dasselbe Ergebnis bei dem folgenden Aufruf erhalten:

LAMBDA [[X] [2 * :X + 1 / LAMBDA [[X] [:X * :X]] 5]] 5

Hier noch ein Beispiel, bei dem die Variable X in den beiden LAMBDA-Aufrufen an verschiedene Werte gebunden wird:

LAMBDA [[X] [:X + LAMBDA [[X] [1 / :X]] 2]] 7
ERGEBNIS: 7.5

Die durch LAMBDA definierten Funktionen lassen sich auch beliebig verketten und potenzieren. Zunächst einige Beispiele zur Verkettung.

ANWENDUNG PRODUKT [LAMBDA [[X] [1 / :X]]]________________
________________ [LAMBDA [[X] [:X * :X]]]   2
ERGEBNIS: 0.25

ANWENDUNG PRODUKT [LAMBDA [[X] [1 / :X]]]   [Q]   2
ERGEBNIS: 0.25

An den letzten beiden Beispielen zeigt sich auch deutlich, daß der Lambda-Ausdruck LAMBDA [[X] [1 / :X]] nicht einfach durch den Term 1 / :X ersetzbar ist, denn im Gegensatz zu den letzten beiden Beispielen wäre die Verkettung der "Funktionen" 1 / :X und Q nicht möglich.

Nun noch einige Beispiele zum Potenzieren (vgl. Abschnitt 6.6.3 und 6.6.4).

ANWENDUNG FPOT [LAMBDA [[X] [:X * :X]]]   3 2
ERGEBNIS: 256

ANWENDUNG FPOT [LAMBDA [[X] [:X * :X]]]   4 3
ERGEBNIS: 43046721

ANWENDUNG FPOT [LAMBDA [[X] [(:X - 1) / :X]]] 3 4711
ERGEBNIS: 4711.0

ANWENDUNG FPOT [LAMBDA [[Z] [1 - :Z]]] 2 30031
ERGEBNIS: 30031

Als Anwendung hiervon wollen wir uns ansehen, wie die Lambda-Notation bei **Summierungsproblemen** eingesetzt werden kann. In der Mathematik summiert man häufig gewisse Terme, zum Beispiel 1/x oder x∗x über einem bestimmten Bereich auf; das heißt, man läßt die Laufvariable x eine Reihe von Werten durchlaufen (zum Beispiel: 1, 2, 3, ..., 10), bildet die jeweiligen Terme (1/1, 1/2, 1/3, 1/4, ..., 1/10) und summiert alle diese Werte auf. Die folgende Funktion SUMME macht es möglich, derartige Summierungsprozesse in Logo nachzuvollziehen.

```
PR SUMME  :FUNKTION  :ANFANG  :ENDE
 WENN  :ANFANG  >  :ENDE  DANN  RUECKGABE  0
 RUECKGABE ( TUE MITLETZTEM  :ANFANG  :FUNKTION )  + ______________
 ______      ( SUMME  :FUNKTION  ( :ANFANG + 1 )  :ENDE )
ENDE
```

Einige Beispiele:

SUMME [LAMBDA [[X] [:X ∗ :X]]] 1 4
ERGEBNIS: 30

SUMME [Q] 1 4
ERGEBNIS: 30

SUMME [LAMBDA [[X] [1 / :X]]] 1 10
ERGEBNIS: 2.92897

Aufgabe 6.18: Schreiben Sie eine Version von SUMME, bei der Sie die Schrittweite als Parameter eingeben können.

Aufgabe 6.19: Schreiben Sie eine neue Version von DIFFQUOT, die Funktionen in der Form von LAMBDA-Ausdrücken verarbeitet. Verwenden Sie die Auswertungsfunktion ANWENDUNG an Stelle von FA.

Um *Rechenausdrücke mit mehreren Variablen* als Funktionen behandeln zu können, müssen wir unsere bisherige LAMBDA-Funktion noch etwas modifizieren. Da die Anzahl der zu verwendenden Variablen nicht von vorn herein festgelegt werden soll, wollen wir alle zu bindenden Parameter jetzt in einer Liste zusammenfassen. Die neue Version möge L.LAMBDA heißen.

Zur Illustration sei noch einmal das Beispiel "Widerstand bei Parallelschaltung" aus Kapitel 2 aufgegriffen. Die dort gegebene Funktion läßt sich in informeller mathematischer Notation folgendermaßen beschreiben:

(R1, R2) —> (R1 ∗ R2) / (R1 + R2)

In der neuen Lambda-Notation soll sie folgendermaßen lauten:

L.LAMBDA [[R1 R2] [:R1 ∗ :R2 / (:R1 + :R2)]]

und so aufgerufen werden:

L.LAMBDA [[R1 R2] [:R1 * :R2 / (:R1 + :R2)]] [50 75]

Die folgende Lambda-Version leistet das Gewünschte:

```
PR L.LAMBDA  :FUNKTION  :WERTELISTE
 LOKAL  ERSTES  :FUNKTION
 VARIABLENBINDUNG  ERSTES  :FUNKTION  :WERTELISTE
 RUECKGABE  TUE  LETZTES  :FUNKTION
ENDE
```

Durch den Aufruf LOKAL ERSTES :FUNKTION werden alle Variablen, die in der ersten Teilliste von :FUNKTION stehen (dies sind genau die zu bindenden Variablen) als lokal relativ zu L.LAMBDA deklariert. (In einer Logo-Version ohne den LOKAL-Befehl kann diese Zeile einfach weggelassen werden; die zu bindenden Variablen werden dann allerdings global).

Die Bindung der Variablen an die in der Werteliste stehenden Werte wird von der Prozedur VARIABLENBINDUNG übernommen.

```
PR VARIABLENBINDUNG  :VARIABLENLISTE  :WERTELISTE
 WENN  :VARIABLENLISTE = [ ]  DANN RUECKKEHR
 WENN  :WERTELISTE  = [ ]  DANN  RUECKKEHR
 SETZE  ERSTES  :VARIABLENLISTE  ERSTES  :WERTELISTE
 VARIABLENBINDUNG  OE  :VARIABLENLISTE  OE  :WERTELISTE
ENDE
```

Einige Beispiele:

L.LAMBDA [[R1 R2] [:R1 * :R2 / (:R1 + :R2)]] [50 75]
ERGEBNIS: 30.0

L.LAMBDA [[A B] [(1 / :A) + (1 / :B)]] [2 3]
ERGEBNIS: 0.833333

ANWENDUNG PRODUKT [LAMBDA [X] [1 / :X]] ______________
______ [L.LAMBDA [[A B] [(1 / :A) + (1 / :B)]]] [50 75]
ERGEBNIS: 30.0 (vgl. vorletztes Beispiel)

L.LAMBDA [[X Y Z] [:X * :X + :Y * :Y = :Z * :Z]] [3 4 5]
ERGEBNIS: WAHR

L.LAMBDA [[X Y Z] [:X * :X + :Y * :Y = :Z * :Z]] [6 7 8]
ERGEBNIS: FALSCH

Aufgabe 6.20: Bei LAMBDA war der zweite Parameter nur ein Wert; bei L.LAMBDA eine Werteliste. Schreiben Sie eine Prozedur
SUPER.LAMBDA :FUNKTION :W ,
die ihrerseits LAMBDA oder L.LAMBDA aktiviert, je nachdem, ob :W ein einzelner Wert oder eine Liste von Werten ist.

Aufgabe 6.21: Birgit Binding hat eine Logo-Version, bei der das Logo-Grundwort LOKAL zu Erzeugung lokaler Variabler als Eingabeparameter nur einen Variablennamen (und nicht eine Liste von Namen) zuläßt. Sie versucht, sich mit der folgen-

den Prozedur LOKAL.BIRGIT zu behelfen, die sie in L.LAMBDA an Stelle des dort verwendeten Grundwortes LOKAL einsetzt.

```
PR LOKAL.BIRGIT  :VARIABLENLISTE
 WENN  :VARIABLENLISTE = [ ]  DANN RUECKKEHR
 LOKAL  ERSTES  :VARIABLENLISTE
 LOKAL.BIRGIT  OHNEERSTES  :VARIABLENLISTE
ENDE
```

Sie hat aber keinen Erfolg damit. Wo liegt der Fehler?

Aufgabe 6.22: Birgits Freundin, Lola Local, hat eine andere Idee. Sie hat schon etwas in diesem Buche vorgeblättert und die SOLANGE-Kontrollstruktur im unmittelbar folgenden Kapitel 7 entdeckt. Sie versucht es folgendermaßen:

```
PR L.LAMBDA.LOLA  :FUNKTION  :WERTELISTE
 LOKAL  "VARIABLENLISTE
 SETZE  "VARIABLENLISTE  ERSTES  :FUNKTION
 SOLANGE [ NICHT  :VARIABLENLISTE = [ ] ] ______________________________
 ______  [ LOKAL  ERSTES  :VARIABLENLISTE ______________________________
 ______    SETZE  ERSTES  :VARIABLENLISTE  ERSTES  :WERTELISTE __________
 ______    SETZE  "VARIABLENLISTE  OHNEERSTES  :VARIABLENLISTE __________
 ______    SETZE  "WERTELISTE  OHNEERSTES  :WERTELISTE  ]
 RUECKGABE  TUE  LETZTES  :FUNKTION
ENDE
```

Aber auch Lolas Versuch führt nicht zum Erfolg. Erklären Sie dies, nachdem Sie sich im nächsten Kapitel mit der SOLANGE-Kontrollstruktur vertraut gemacht haben.

Aufgabe 6.23: Schließlich gelingt es Birgits Klassenkameradin, Eva Lu Ator, das Problem mit Hilfe des GEHE-Befehls zu lösen. Wie könnte Evas Lösung aussehen?

Kapitel 7: Programmieren als Spracherweiterung

Wir haben bisher eine ganze Reihe von Programmen geschrieben, obwohl wir sie meist nicht als solche bezeichnet haben. Stattdessen haben wir von Prozeduren oder Funktionen gesprochen. Dennoch dienten diese Prozeduren ähnlichen Zielen, wie sie durch klassische Programmierung verfolgt werden. Man denke etwa an die Prozedur zur Erstellung von Tilgungsplänen in Kapitel 6 (Abschnitt 6.1 und 6.3).

Im Stil der Programmierung hat sich unsere Methode jedoch deutlich von den Methoden der klassischen Programmierung unterschieden. Anstatt ein großes Mammutprogramm in Angriff zu nehmen, haben wir lauter kleine, überschaubare Moduln geschrieben, die selbständig verwendbar waren. Jede Logo-Prozedur konnte nach ihrer Definition genau wie ein Logo-Grundbefehl verwendet und somit auch sofort nach ihrer Erstellung ausgetestet werden. Durch die Definition von Prozeduren und Funktionen haben wir das Logo-Grundsystem kontinuierlich durch neue Superbefehle und Super-Superbefehle angereichert, bis wir unser Ziel jeweils durch Aufruf eines einzigen Prozedurnamens erreichen konnten.

Anstatt Programme im klassischen Sinne zu schreiben, haben wir das Logo-Grundsystem durch neue Befehle erweitert. Bei dem elementaren Charakter der bisherigen Einführungsbeispiele ist die Tragweite dieser Methode zwar nur ansatzweise zum Zuge gekommen; der Sachverhalt ist bei umfangreicheren Projekten jedoch prinzipiell derselbe. Eine weitere Programmiersprache, die auf dem Prinzip des Programmierens durch Spracherweiterung aufgebaut ist, ist (natürlich neben Lisp, der Muttersprache von Logo) die Sprache Forth.

Daß man eine Sprache durch *Moduln* erweitert, ist jedoch kein absolut neuer Gedanke, der erst mit Lisp/Logo bekannt geworden ist. In anderen Sprachen hat man durch die Erstellung von Modul-Bibliotheken ähnliche Ziele verfolgt. Neu in Lisp/Logo ist jedoch im Vergleich zu den sonst meist sehr schwerfälligen Methoden des Einbindens von Bestandteilen dieser Modul-Bibliotheken in das jeweils gerade zu erstellende Programm die Unkompliziertheit, mit der neue Moduln mit den schon bestehenden Moduln interagieren können.

Neu ist außerdem die umfassende Art und Weise, in der Spracherweiterungen möglich sind. Man kann in Lisp/Logo nicht nur neue Prozeduren oder Funktionen im klassischen Sinne schreiben, sondern man kann dem Grundsystem auch **neue Kontrollstrukturen** und **neue Datenstrukturen** hinzufügen. Mit diesem Thema wollen wir uns im folgenden eingehender befassen.

Alle im folgenden diskutierten Kontrollstrukturen sind endrekursiv. Sie können deshalb beliebig lange laufen.

7.1 Wiederholungen: Die SOLANGE (WHILE) Kontrollstruktur

Die moderneren Sprachen der Algol-Familie (besonders Pascal) zeichnen sich durch ein vergleichsweise reichhaltiges Repertoire an Kontrollstrukturen aus. Eine der wichtigsten Kontrollstrukturen zur Kontrolle von schleifenartigen Abläufen ist SOLANGE (englisch: WHILE). Zur Logik der Verwendung von SOLANGE gehört die Angabe einer Bedingung und einer Handlung. Solange die genannte Bedingung erfüllt ist, wird die spezifizierte Handlung ausgeführt.

SOLANGE ist zwar nicht im Logo-Grundsystem enthalten, wir können es aber leicht selbst definieren. Bei allen diesen neu einzuführenden Kontrollstrukturen spielt der listenauswertende TUE-Grundbefehl eine entscheidende Rolle. Es ist deshalb sehr natürlich, die Bedingung und die Handlung zum Zwecke der Auswertung durch den TUE-Befehl als Listen zu schreiben.

```
PR SOLANGE  :BEDINGUNG  :HANDLUNG
 PRUEFE  TUE  :BEDINGUNG
 WENNFALSCH DANN RUECKKEHR
 TUE  :HANDLUNG
 SOLANGE  :BEDINGUNG  :HANDLUNG
ENDE
```

Der Befehl TUE :BEDINGUNG erzwingt die Auswertung der Bedingungsliste; der entsprechende Wahrheitswert wird von PRUEFE registriert. Entsprechend erzwingt TUE :HANDLUNG die Ausführung dessen, was in der Handlungsliste steht. Die vorletzte Zeile zeigt, daß SOLANGE eine endrekursive Prozedur ist und deshalb beliebig lange laufen kann. Auch dies ist wieder ein Beispiel dafür, daß die Rekursion als eine der fundamentalen Kontrollstrukturen von Lisp/Logo anzusehen ist, aus der sich andere Kontrollstrukturen aufbauen lassen.

Eine gleichwertige Formulierung für die SOLANGE-Kontrollstruktur ist:

```
PR  SOLANGE  :BEDINGUNG  :HANDLUNG
 WENN  TUE  :BEDINGUNG  DANN  TUE  :HANDLUNG  SONST  RUECKKEHR
 SOLANGE  :BEDINGUNG  :HANDLUNG
ENDE
```

SOLANGE läßt sich im Direktausführungsmodus und von anderen Prozeduren aus verwenden. Letzteres haben wir zum Beispiel schon im Programm TILGUNGSPLAN in Kapitel 6 kennengelernt. Hier noch ein weiteres Beispiel im Direktausführungsmodus.

```
SETZE "X 10
SOLANGE [ :X < 15 ] [ ( DZ :X LEERWORT :X*:X*:X ) SETZE "X :X+1 ]

10    1000
11    1331
12    1728
13    2197
14    2744
```

(Die Funktion LEERWORT wurde in Kapitel 4, Abschnitt 4.2.4 behandelt).

Aus anderen Sprachen ist man gelegentlich gewöhnt, daß solche Kontrollstrukturen durch ein "Endwort" abgeschlossen werden. Auch dies läßt sich, wenn man es unbedingt haben will, leicht in Logo realisieren:

```
PR ENDE.SOLANGE
ENDE
```

ENDE.SOLANGE ist die leere Prozedur, die überhaupt nichts tut.

Ein Beispiel:

```
SETZE "X 1
SOLANGE [ ABS( 5 - :X * :X ) > 0.001 ]
        [ SETZE "X (:X + 5 / :X ) / 2
          DRUCKEZEILE :X ] ENDE.SOLANGE
```

(ABS ist die Kurzbezeichnung der Funktion ABSOLUTBETRAG aus Kapitel 2, Abschnitt 2.4.3). Ein Probelauf dieser Anweisungsfolge ergibt:

3.0
2.33333
2.23809
2.23607

Probe:
2.23607 * 2.23607
ERGEBNIS: 5.0

Die obige Anweisungsfolge ist ein Beispiel für das sogenannte **Verfahren von Heron** zur Ermittlung von Quadratwurzeln.

Aufgabe 7.1: (a) Schreiben Sie die obige Anweisungsfolge als Prozedur.
(b) Informieren Sie sich in einem geeigneten Mathematikbuch über das Heron-Verfahren.
(c) Schreiben Sie eine Funktion HERON :A zu Ermittlung der Quadratwurzel von :A.
(d) Verallgemeinern Sie die Funktion aus (c) zur Funktion HERON :K :A , welche die K-te Wurzel von :A berechnet.

Schauen Sie jetzt nochmals Aufgabe 6.22 an.

7.2 Wiederholungen: Die WDH (REPEAT) Kontrollstruktur

Bei der SOLANGE-Kontrollstruktur kann es vorkommen, daß die angegebene Handlung überhaupt nicht ausgeführt wird; nämlich dann, wenn die Bedingung schon beim Eintritt in die SOLANGE-Schleife nicht erfüllt ist. Mit den früher stärker gebräuchlichen Flußdiagrammen kann man die SOLANGE-Kontrollstruktur folgendermaßen darstellen:

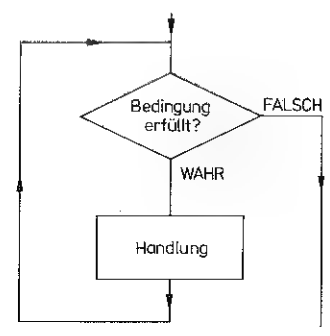


Abbildung 7.1: "solange ... " im Flußdiagramm

Mit der in SOLANGE vorkommenden Bedingung hat man die Kontrolle darüber, ob die angegebene Handlung überhaupt auszuführen ist. Es ist also eine Ausführungs-Bedingung. Manchmal möchte man jedoch, daß zunächst die die angegebene Handlung ausgeführt und dann geprüft wird, ob dies fortgesetzt oder ob damit Schluß gemacht werden soll. In diesem Fall handelt es sich um eine Abbruchs-Bedingung. Im Flußdiagramm sieht das dann so aus:

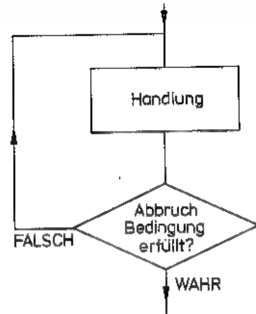


Abbildung 7.2: "wiederhole ... bis ... " im Flußdiagramm

Die Wiederholungs-Kontrollstruktur wird meist in der folgenden syntaktischen Form verwendet:

WIEDERHOLE Handlung BIS Bedingung.

In Pascal: REPEAT statement(s) UNTIL expression.

Wenn wir eine solche Kontrollstruktur in Logo nachbauen wollen, dann können wir natürlich nicht das schon vergebene Grundwort WIEDERHOLE (bzw. WH) verwenden. Wir nennen die neue Kontrollstruktur deshalb kurz WDH. Von der Diskussion der SOLANGE-Kontrollstruktur her wissen wir, daß die Handlung und die Bedingung am besten in Form von Listen zu schreiben sind. Daraus ergibt sich, daß ein BIS eigentlich unnötig ist.

```
PR WDH  :HANDLUNG  :BEDINGUNG
  TUE  :HANDLUNG
  PRUEFE  TUE  :BEDINGUNG
  WENNWAHR  RUECKKEHR
  WDH  :HANDLUNG  :BEDINGUNG
ENDE
```

Ein Beispiel für die Verwendung von WDH:

```
PR ZAHLENRATEN  :MAXIMUM  :TOLERANZ
 LOKAL  "GEHEIMZAHL
 LOKAL  "SCHAETZWERT
 SETZE  "GEHEIMZAHL  ZUFALLSZAHL  :MAXIMUM
 WDH  [ DRUCKE  [ IHR SCHAETZWERT: ]
        SETZE  "SCHAETZWERT  ERSTES  EINGABE
        AUSWERTUNG  :SCHAETZWERT  :GEHEIMZAHL  ]
      [ NICHT  ABS  ( :SCHAETZWERT  -  :GEHEIMZAHL  )  >  :TOLERANZ  ]
ENDE
```

```
PR AUSWERTUNG  :SCHAETZWERT  :GEHEIMZAHL
 WENN  :SCHAETZWERT  =  :GEHEIMZAHL  DZ [ STIMMT GENAU ! ] ____________
 _____ SONST ______________________________________________________________
 _____ WENN NICHT ABS (:SCHAETZWERT - :GEHEIMZAHL )  >  :TOLERANZ ______
 ________  DZ  [ IM TOLERANZBEREICH ] _________________________________
 ________  SONST  WENN  :SCHAETZWERT  <  :GEHEIMZAHL ________________
 _____________  DZ  [ ZU KLEIN ] ______________________________________
 _____________  SONST DZ  [ ZU GROSS ]
ENDE
```

Wer auf das BIS (bzw. UNTIL) absolut nicht verzichten möchte, kann es natürlich gern haben:

```
PR BIS  :B
 RUECKGABE  :B
ENDE
```

Die Funktion BIS leitet also nur ihr Argument weiter. Ein kleines Beispiel:

```
SETZE "X 100
WDH [ ( DZ :X LEERWORT :X * :X ) SETZE "X :X+1 ] BIS [ :X = 105 ]
```

```
100    10000
101    10201
102    10404
103    10609
104    10816
```

Aufgabe 7.2: Schreiben Sie (entsprechend der zweiten Form von SOLANGE) mit Hilfe von WENN ... DANN ... SONST ... eine Version von WDH, die aus weniger Zeilen besteht.

7.3 Fallunterscheidungen: die FALL (CASE) Kontrollstruktur

Die Verwendung soll an einem kleinen Beispiel erläutert werden: Bei bestimmten Computeranwendungen muß man die Wochentage von bestimmten Ereignissen speichern, um sie später weiterverarbeiten zu können. Man speichert die Wochentage aber meist nicht "wörtlich" in Form ihres Namens sondern in einer codierten Form, denn diese kann viel kürzer gewählt werden; zum Beispiel folgendermaßen:

0: Sonntag
1: Montag
2: Dienstag
3: Mittwoch
4: Donnerstag
5: Freitag
6: Sonnabend

In Ausdrucken möchte man aber der Lesbarkeit halber die Wochentage nicht als Codenummer sondern voll ausgeschrieben haben. Man könnte hierfür eine kleine Decodierungs-Prozedur etwa im folgenden Stile schreiben:

```
PR DECODIERUNG.WOCHENTAG  :N
 WENN  :N = 0  DZ  "SONNTAG ________________________________________
 ___  SONST WENN  :N = 1 DZ  "MONTAG ________________________________
 _____________ SONST ...
 ...
ENDE
```

Lange WENN ... DANN ... SONST ... - Ausdrücke werden schnell unübersichtlich, besonders, wenn sie noch kompliziert verschachtelt sind. Manche Programmiersprachen verfügen deshalb über eine FALL (CASE) - Kontrollstruktur, mit deren Hilfe in bestimmten Fällen solche langen WENN ... DANN ... SONST ... - Bandwürmer vermieden werden können.

In Pascal könnte man die Wochentags-Prozedur zum Beispiel folgendermaßen schreiben:

```
PROCEDURE DECODIERUNGWOCHENTAG(N: INTEGER);
BEGIN
  CASE N OF
    0: WRITE('SONNTAG');
    1: WRITE('MONTAG');
      ...
    6: WRITE('SONNABEND')
  END
END;
```

Eine derartige Kontrollstruktur, wir wollen sie im folgenden FALL nennen, muß also über die folgenden Parameter verfügen:

- ein Identifikationsobjekt (im obigen Beispiel N);
- eine Auswahlliste.

In der Auswahlliste müssen alle möglichen Werte für das Identifikationsobjekt stehen; und zu jedem Wert muß eine Handlung angegeben sein, die auszuführen ist, wenn das Identifikationsobjekt diesen Wert annimmt. Im obigen Beispiel könnte die Auswahlliste zum Beispiel so aussehen:

```
WERT AUSWAHLLISTE =

[ [ 0 [ DRUCKE "SONNTAG    ] ]
  [ 1 [ DRUCKE "MONTAG     ] ]
  [ 2 [ DRUCKE "DIENSTAG   ] ]
  ...
  [ 6 [ DRUCKE "SONNABEND ] ] ]
```

Wie in diesem Beispiel wollen wir die Auswahlliste im folgenden als Liste von Listen organisieren. Jede Teilliste der Auswahlliste besteht aus zwei Komponenten:

einem Wert und einer in Form einer Liste angegebenen Handlung. Wenn das Identifikationsobjekt den Wert annimmt, ist die entsprechende Handlung auszuführen.

Bei der CASE-Kontrollstruktur soll üblicherweise höchstens eine Variante "zum Zuge kommen". Wenn also eine Übereinstimmung festgestellt und die dazugehörige Handlung ausgeführt worden ist, wird die FALL-Prozedur verlassen.

```
PR FALL  :IDENTIFIKATIONSOBJEKT  :AUSWAHLLISTE
 WENN  :AUSWAHLLISTE = [ ] DANN  RUECKKEHR
 WENN  :IDENTIFIKATIONSOBJEKT  =  ERSTES  ERSTES  :AUSWAHLLISTE _________
 _____  DANN  TUE  LETZTES  ERSTES  :AUSWAHLLISTE  RUECKKEHR
 FALL  :IDENTIFIKATIONSOBJEKT OHNEERSTES :AUSWAHLLISTE
ENDE
```

Die Wochentags-Prozedur sieht nun folgendermaßen aus:

```
PR DECODIERUNG.WOCHENTAG  :N
 FALL :N  [ [ 0  [ DRUCKE  "SONNTAG    ] ] ____________________
 ______     [ 1  [ DRUCKE  "MONTAG     ] ] ____________________
 ______     [ 2  [ DRUCKE  "DIENSTAG   ] ] ____________________
 ______     [ 3  [ DRUCKE  "MITTWOCH   ] ] ____________________
 ______     [ 4  [ DRUCKE  "DONNERSTAG ] ] ____________________
 ______     [ 5  [ DRUCKE  "FREITAG    ] ] ____________________
 ______     [ 6  [ DRUCKE  "SONNABEND  ] ] ]
ENDE
```

In vielen Algol-ähnlichen Sprachen ist die CASE-Kontrollstruktur nach dem Muster der obigen Prozedur FALL verwirklicht. Diese Version enthält jedoch noch eine Schwachstelle. Wenn wir zum Beispiel die Prozedur DECODIERUNG.WOCHENTAG mit dem Eingabeparameter 7 aufrufen, tut sich gar nichts. Es wäre aber häufig wünschenswert, auch in Fällen, wo das Identifikationsobjekt keinen der angegebenen Werte annimmt, eine bestimmte Handlung, zum Beispiel eine Fehlermeldung, angeben zu können.

In der Prozedur DECODIERUNG.WOCHENTAG wäre es zum Beispiel gut, wenn als letztes Element der Auswahlliste die folgende Teilliste eingefügt werden könnte:

[ANSONSTEN [DRUCKE [FALSCHE EINGABE]]]

Die folgende verbesserte Version von FALL trägt diesem Aspekt Rechnung:

```
PR FALL  :IDENTIFIKATIONSOBJEKT  :AUSWAHLLISTE
 WENN  :AUSWAHLLISTE = [ ]   RUECKKEHR
 WENN  ERSTES  ERSTES  :AUSWAHLLISTE  =  "ANSONSTEN ___________
 _____  TUE  LETZTES  ERSTES  :AUSWAHLLISTE  RUECKKEHR
 WENN  :IDENTIFIKATIONSOBJEKT  =  ERSTES  ERSTES  :AUSWAHLLISTE _________
 ______  TUE  LETZTES  ERSTES  :AUSWAHLLISTE  RUECKKEHR
 FALL :IDENTIFIKATIONSOBJEKT OHNEERSTES :AUSWAHLLISTE
ENDE
```

Dies ist der Vorteil von erweiterbaren Sprachen. Man muß nicht die (n + 1)-te verbesserte Version des Compilers abwarten und für teures Geld erstehen, sondern man kann die Sprache direkt seinen Wünschen anpassen.

Als aufmerksamer Leser haben Sie sich bei der zweiten Version von FALL möglicherweise gefragt, warum das Objekt ERSTES ERSTES :AUSWAHLLISTE nicht als Prozedurname interpretiert wird, wenn es gleich ANSONSTEN ist. Das hängt damit zusammen, wie Listen interpretiert werden. Logo interpretiert Worte, die in einer Liste vorkommen so als ob sie in gequoteter Form vorlägen (siehe Abschnitt 4.3.2.1).

7.4 Zählschleifen: Die FUER (FOR) Kontrollstruktur

Das Muster der Verwendung der FOR-Kontrollstruktur verläuft in den meisten Sprachen etwa folgendermaßen:

```
FOR X = 10 TO 100 STEP 5 DO Handlung.
```

Natürliche Parameter einer in Logo nachzubildenden FUER-Kontrollstruktur sind also: der Name der Laufvariablen, der Anfangswert, der Endwert, die Schrittweite und die auszuführende Handlung. In Pascal darf die Laufvariable nur ganzzahlige Werte annehmen und es sind nur Schrittweiten vom Betrage Eins erlaubt. Wir wollen hier jedoch auf diese von vielen Benutzern als störend empfundenen Einschränkungen verzichten.

```
PR FUER  :LAUFVARIABLE  :ANFANG  :ENDE  :SCHRITT  :HANDLUNG
 LOKAL  :LAUFVARIABLE
 SETZE  :LAUFVARIABLE  :ANFANG
 WENN  WERT  :LAUFVARIABLE  >  :ENDE  DANN  RUECKKEHR
 TUE  :HANDLUNG
 FUER  :LAUFVARIABLE  (:ANFANG + :SCHRITT)  :ENDE  :SCHRITT  :HANDLUNG
ENDE
```

In diesem Beispiel ist :LAUFVARIABLE ein Eingabeparameter, der den *Namen* der Laufvariablen zugewiesen bekommen soll. WERT :LAUFVARIABLE ist der (numerische) Wert dieser letzteren Variablen. Im folgenden Beispiel ist :LAUFVARIABLE = K; WERT :LAUFVARIABLE, d.h. :K, variiert also zwischen 0 und 10. Auch diese Prozedur ist wieder endrekursiv. Ein Beispiel, wie sie zu verwenden ist:

```
FUER "K 0 10 2  [(DRUCKEZEILE :K LEERWORT :K * :K * :K * :K)]

0     0
2     16
4     256
6     1296
8     4096
10    10000
```

Die Prozedur FUER beruht wieder entscheidend auf dem Variablenkonzept von Logo. LAUFVARIABLE ist der Name einer Variablen, deren Wert ein weiterer Variablenname ist. Durch den Aufruf von FUER wird die folgende Privatbibliothek erzeugt:

Privatbibliothek von FUER

Variablenname	Wert der Variablen
LAUFVARIABLE	K
K	0

In Logo-Versionen, die nicht über lokale Variable verfügen, könnte man die Laufvariable folgendermaßen zunächst als globale Variable erzeugen und nach Ablauf der Zählschleife löschen:

```
PR FUER  :LAUFVARIABLE  :ANFANG  :ENDE  :SCHRITT  :HANDLUNG
 SETZE  :LAUFVARIABLE  :ANFANG
 WENN  WERT  :LAUFVARIABLE  >  :ENDE  ________________
 ____  DANN  VERGISSNAME  :LAUFVARIABLE  RUECKKEHR
 TUE  :HANDLUNG
 FUER  :LAUFVARIABLE  (:ANFANG + :SCHRITT)  :ENDE  :SCHRITT  :HANDLUNG
ENDE
```

Aufgabe 7.3: In der letzten Version von FUER kann es passieren, daß als Laufvariable ein Name verwendet wird, der schon als globale Variable existiert und mit einem bestimmten Wert belegt ist. Durch den Aufruf von FUER würde dieser Wert überschrieben. Ändern Sie die zweite Version von FUER so ab, daß ein solcher Wert gerettet und nach Ablauf von FUER wieder der entsprechenden globalen Variablen zugewiesen wird.

7.4.1 Eine "intelligentere" Version von FUER

In sehr vielen Anwendungen der FUER-Kontrollstruktur ist die Schrittweite Eins. In diesen Fällen ist es lästig, diesen Wert immer als aktuellen Parameter übergeben zu müssen. Es wäre bequemer, dann nur

FUER :LAUFVARIABLE :ANFANG :ENDE :HANDLUNG

zu schreiben.

Auch ein weiterer Punkt könnte noch etwas benutzerfreundlicher gestaltet werden. Wenn der Endwert unter dem Anfangswert liegt, muß die Schrittweite natürlich negativ sein, damit man sich überhaupt auf den Endwert zubewegt. Dennoch stellt sich der Benutzer die Schrittweite oft als positive Größe vor; er operiert dann also mit dem Absolutwert der Schrittweite. Eine "intelligentere" Version von FUER sollte im gewissen Umfang "erkennen" können, was der Benutzer wohl gemeint hat, selbst wenn er die Schrittweite fälschlicherweise als positive Zahl eingegeben hat.

Die folgende Version von FUER trägt diesen Gesichtspunkten Rechnung. Da die Anzahl der Parameter einer Prozedur festliegt, kann man den Parameter Schrittweite nicht einfach gelegentlich hinzufügen und gelegentlich weglassen. Deswegen ist die Parametereingabe in der neuen Version grundsätzlich anders gestaltet worden als in der obigen Version. Es gibt nur noch einen einzigen Eingabeparameter, eine Liste, die alle bisherigen Parameter als Elemente enthält. Je nachdem, ob die Liste fünf oder nur vier Elemente enthält, wird die Schrittweite übernommen oder automatisch auf Eins gesetzt. Die Handlung, die schon oben als Liste formuliert war, ist jetzt natürlich als Teilliste der Parameterliste zu schreiben.

```
PR FUER  :PARAMETERLISTE
 LOKAL  ERSTES  :PARAMETERLISTE
 LOKAL  "SCHRITT
 PRUEFE  ( ELEMENTEZAHL  :PARAMETERLISTE )  =  5
 WENNWAHR  DANN  SETZE  "SCHRITT  VIERTES  :PARAMETERLISTE
 WENNFALSCH  DANN  SETZE  "SCHRITT  1
 PRUEFE  ( ZWEITES  :PARAMETERLISTE ) > ( DRITTES  :PARAMETERLISTE )
 WENNWAHR DANN WENN :SCHRITT > 0  DANN SETZE  "SCHRITT  (-1)*:SCHRITT
 FUER.1  (ERSTES  :PARAMETERLISTE)  (ZWEITES  :PARAMETERLISTE) ________
 ______  (DRITTES :PARAMETERLISTE)  :SCHRITT  (LETZTES :PARAMETERLISTE)
ENDE
```

```
PR FUER.1  :VARIABLENNAME  :ANFANG  :ENDE  :SCHRITT  :HANDLUNG
 PRUEFE  :SCHRITT > 0
 WENNWAHR  DANN  WENN  :ANFANG  >  :ENDE  DANN  RUECKKEHR
 WENNFALSCH  DANN  WENN  :ANFANG  <  :ENDE  DANN  RUECKKEHR
 SETZE  :VARIABLENNAME  :ANFANG
 TUE  :HANDLUNG
 FUER.1  :VARIABLENNAME (:ANFANG + :SCHRITT) :ENDE :SCHRITT :HANDLUNG
ENDE
```

Einige Beispiele (im Hinblick auf die Problematik der stellengerechten Darstellung sei auf die Hinweise zu Beginn des Buches verwiesen):

```
FUER [ I 1 5 [ ( DRUCKEZEILE :I LEERWORT :I * :I ) ] ]

1    1
2    4
3    9
4    16
5    25
```

```
FUER [ X 0 1 0.2 [ ( DRUCKEZEILE :X LEERWORT SIN :X ) ] ]

0           0
0.2         0.00348963
0.4         0.00698032
0.6         0.010471
0.799999    0.0139617
0.999999    0.0174513
```

Dieses Beispiel zeigt, daß es problematisch ist, **Gleitkommazahlen als Schrittweiten** zuzulassen. Denn diese Zahlen werden im Gegensatz zu den Ganzzahlvariablen computerintern nur näherungsweise als Binärzahlen gespeichert. Alle Rechnungen mit Gleitkommazahlen werden in dieser Binärdarstellung durchgeführt; nur die Ergebnisse werden dann wieder (automatisch) vom Computer als Dezimalzahlen dargestellt. Bei dieser Umwandlung ins Zweiersystem und der Rückwandlung ins Zehnersystem treten nun prinzipiell unvermeidbare Näherungsfehler auf, die zu solchen Resultaten wie dem obigen Ausdruck führen.

Man kann dieses Problem auf zwei Weisen angehen. In Pascal wird zum Beispiel

gesagt: "Dies ist ein völlig unakzeptabler Zustand, der gar nicht entstehen darf. Er wird auf syntaktischer Ebene unterbunden. Die Schrittvariable darf nur vom Typ einer ganzen Zahl sein." In anderen Sprachen stellt man sich auf den Standpunkt: "Dieser Sachverhalt ist zwar unerfreulich; dennoch gibt es viele Situationen, wie zum Beispiel in den meisten naturwissenschaftlichen Anwendungen, wo man die Schrittweite klein (auf jeden Fall betragsmäßig kleiner als Eins) halten möchte. Wir gehen davon aus, daß wir es mit einem mündigen Benutzer zu tun haben, dem die Problematik der Gleitkommazahlen bekannt ist und der in angemessener Weise auf das obige Problem zu reagieren weiß."

Weitere Beispiele:

```
FUER [ K 20 16 [ ( DRUCKEZEILE :K LEERWORT :K * :K * :K ) ] ]

20      8000
19      6859
18      5832
17      4913
16      4096

FUER [ R 100 98 0.5 [ ( DRUCKEZEILE :R LEERWORT QW :R ) ] ]

100     10
99.5     9.97497
99       9.94987
98.5     9.92471
98       9.89949

FUER [ T 4 2.5 (-0.25) [ ( DRUCKEZEILE :T LEERWORT COS :T ) ] ]

4       0.997564
3.75    0.99783
3.5     0.998097
3.25    0.998363
3       0.998629
2.75    0.99882
2.5     0.99901
```

Die beiden Versionen von FUER eignen sich gut, um nochmals auf die Art und Weise hinzuweisen, wie Logo den Inhalt von Listen analysiert (vergleiche Abschnitt 4.3.2.1). Die einfache Form von FUER mußte folgendermaßen aufgerufen werden:

```
FUER "K 0 10 2 [ ( DRUCKEZEILE :K LEERWORT :K * :K * :K * :K ) ]
```

In der intelligenteren Version lautet der Aufruf dagegen so:

```
FUER [ K 0 10 2 [ ( DRUCKEZEILE :K LEERWORT :K * :K * :K * :K ) ] ]
```

Der Name der Laufvariablen (hier K) darf nicht gequotet werden, wenn er in einer Liste steht. Denn Worte, die in einer Liste vorkommen, werden von Logo automatisch so interpretiert als seien sie gequotet.

7.5 Neue Datenstrukturen: FELD (ARRAY) und MATRIX

In vielen mathematischen, naturwissenschaftlichen oder wirtschaftswissenschaftlichen Anwendungen arbeitet man gern mit einer Datenstruktur, die in der Informatik als FELD (englisch: ARRAY) bezeichnet wird. Ein Feld ist eine Aneinanderreihung einer bestimmten Anzahl von gleichartigen Elementen. In mathematischer Sprechweise wird es als n-Tupel oder Vektor bezeichnet. Felder, deren Elemente wieder Felder sind, nennt man auch Matrizen.

Zum Umgang mit Feldern gehören die folgenden Grundoperationen:

- Definition mit Dimensionierung eines Feldes;
- Bestückung eines Feldes mit Anfangswerten;
- Bestückung einzelner (durch Indizes bezeichneter) Feldelemente mit Werten;
- Zugriff zu den einzelnen Feldelementen durch (ganzzahlige) Indizes.

Die ersten beiden Operationen (das heißt: Definition, Dimensionierung und Bestükkung mit Anfangswerten) werden im folgenden auch als Initialisierung eines Feldes bezeichnet.

In mathematischer Syntax wird das I-te Element des Feldes A meist als A(I) geschrieben. In Logo wird es syntaktisch etwas einfacher, wenn wir an Stelle von A(I) die Schreibweise A.I verwenden. An der Sache selber ändert sich dadurch nichts. Der Wertebereich, den die Variable I durchläuft, nennen wir im folgenden den Indexbereich (englisch: subscript range) des Feldes; ihr höchster Wert gibt die Komponentenzahl des Feldes wieder.

Die folgende Prozedur DEF.FELD übernimmt die Initialisierung eines Feldes. Wir verwenden dabei die intelligentere Version von FUER aus dem vorigen Abschnitt.

```
PR DEF.FELD :FELDNAME :KOMPONENTENZAHL
 FUER ( LISTE "I 1 :KOMPONENTENZAHL ________________________
 ________________ [ SETZE ( WORT :FELDNAME ". :I ) 0 ] )
ENDE
```

Durch den Befehl DEF.FELD "A 6 wird zum Beispiel ein Feld mit Namen A definiert, dessen 6 Elemente als A.1, A.2, ..., A.6 ansprechbar sind. Diese Feldelemente werden im Prozeß der Definition zugleich auf Null gesetzt. Der Zugriff auf die Komponenten ist einfach die Wertrückgabe.

Ein Beispiel:

WERT "A.3 (beziehungsweise :A.3)
ERGEBNIS: 0

Hier ist eine kleine Übersetzungstabelle zwischen der mathematischen Bezeichnungsweise für Felder und Operationen mit Feldern und den entsprechenden Operationen in Logo:

	mathematische Notation	Logo-Notation
Name:	A(I)	A.I
Wert:	A(I)	WERT "A.I oder :A.I
Bestückung:	A(I) := 3.14	SETZE "A.I 3.14

Aufgabe 7.4: Siegfried Schnellhacker hält die obige Definition von DEF.FELD für unnötig umständlich. Er meint, die folgende Version müßte es auch tun:

```
PR DEF.FELD.SIEGFRIED  :FELDNAME  :KOMPONENTENZAHL
 FUER [ I 1 :KOMPONENTENZAHL  [ SETZE ( WORT  :FELDNAME  ".  :I ) 0 ] ]
ENDE
```

Warum funktioniert Siegfrieds Version nicht ?

Der Bequemlichkeit und besseren Lesbarkeit halber führen wir noch die beiden folgenden Hilfsfunktionen zum Bestücken und Lesen von Feldelementen ein:

```
PR BESTUECKE  :FELDNAME  :INDEX  :WERT
 SETZE ( WORT  :FELDNAME  ".  :INDEX )  :WERT
ENDE

PR KOMPONENTE  :FELDNAME  :INDEX
 RUECKGABE  WERT  ( WORT  :FELDNAME  ".  :INDEX )
ENDE
```

Einige Beispiele:
DEF.FELD "B 5
BESTUECKE "B 3 6.28
KOMPONENTE "B 3
ERGEBNIS: 6.28
WERT "B.3
ERGEBNIS: 6.28
:B.3
ERGEBNIS: 6.28

Aufgabe 7.5: Schreiben Sie eine Prozedur zur Definition von Feldern, deren Indexbereich nicht notwendigerweise bei 1 beginnt. (Wie groß ist dann die Komponentenzahl des Feldes?)

Eine Matrix könnte nun zum Beispiel folgendermaßen definiert und initialisiert werden:

```
PR DEF.MATRIX  :NAME  :ZEILENZAHL  :SPALTENZAHL
 FUER ( LISTE  "J  1  :ZEILENZAHL ______________________________
 ________      [ DEF.FELD  (  WORT  :NAME   ".  :J )  :SPALTENZAHL ] )
ENDE
```

Ist der Name der Matrix zum Beispiel M und der aktuelle Wert von J gleich 5, so wird durch die Anweisung (Wort :Name ". :J) der Feldname M.5 "zusammengebaut".

Die Matrizenelemente heißen nun M.1.1, M.1.2, M.1.3, ... , M.2.1, M.2.2, M.2.3, ...

Man könnte sich nun eine ganze Programmierumgebung zur Matrixalgebra erstellen. Dem interessierten Leser mit den entsprechenden mathematischen Vorkenntnissen sei empfohlen, dies zur Übung tatsächlich einmal (zumindest in den Anfangsgründen) zu tun. Man erkennt allerdings bald, daß dies nur eine prinzipielle Lösung ist; denn die selbstgebaute Matrixalgebra wird so langsam, daß man sie in realistischen Situationen wohl kaum benutzen wird. Damit ist jedoch nicht gesagt, daß diese Übungen wertlos seien. Sie können durchaus viel zum Verständnis sowohl der Matrixalgebra als auch von Logo beitragen.

Es wäre allerdings schön, wenn Logo über eine "eingebaute" schnelle Matrixalgebra verfügen würde. Bei neueren Logo-Versionen auf 32-Bit-Rechnern ist dies häufig der Fall.

Daß Felder aber auch unabhängig von der Matrixalgebra nützlich sein können, soll das folgende Simulationsbeispiel zeigen.

7.6 Eine Anwendung von Feldern: Simulation von Zufallsprozessen

Als Beispiel für eine kleine stochastische Simulation, das heißt also für die Simulation eines Zufallsprozesses, wollen wir das **Sammlerproblem** behandeln: Ein Kind sammelt Wertmarken ("Pennies"), die zum Beispiel in Kakaopackungen enthalten sind. Es gibt die Marken von 1 bis 6, die in zufälliger Weise auf die Packungen verteilt sein mögen. Die Marken 1, ... , 6 sollen alle gleichhäufig vorkommen. Wieviele Packungen muß man durchschnittlich kaufen, um einen vollständigen Satz aus allen sechs Marken zu erhalten?

In einem anderen Zusammenhang kann man auch die folgende mathematisch gleichwertige Frage stellen: wie lange muß man durchschnittlich würfeln bis man (erstmals) alle sechs Möglichkeiten erwürfelt hat?

Wir simulieren das Würfeln mit Hilfe von Zufallszahlen.

```
PR WUERFELZAHL
 RUECKGABE 1 + ZUFALLSZAHL 6
ENDE
```

Durch den Aufruf ZUFALLSZAHL 6 wird in der Prozedur WUERFELZAHL in zufälliger Weise eine der Zahlen 0, 1, 2, 3, 4, 5 erzeugt. Durch die Addition von 1 erhalten wir gerade die möglichen "Würfelzahlen".

Einige Beispiele:

```
WUERFELZAHL
ERGEBNIS: 4

WH 5 [ DZ WUERFELZAHL ]
2
5
3
4
3
```

Für die möglichen Würfelergebnisse definieren wir unter dem Namen A ein Feld mit den 6 Komponenten A.1, ... , A.6. Durch den Initialisierungsprozess werden alle Feldelemente auf Null gesetzt. Wird eine bestimmte Zahl, zum Beispiel die 5, gewürfelt, so wird der Inhalt von A.5 um Eins erhöht. Wenn keines der Felder A.I (I = 1,...,6) mehr gleich Null ist, hat man einen vollständigen Satz. Alle Felder A.I sind genau dann von Null verschieden, wenn ihr Produkt von Null verschieden ist.

```
PR WARTEZEIT
 LOKAL  "W     SETZE  "W  0
 LOKAL  "R     SETZE  "R  0
 DEF.FELD  "A  6
 SOLANGE  [ KEIN.VOLLSTAENDIGER.SATZ? ] _____________________
 _______  [ SETZE  "R  WUERFELZAHL      _____________________
 _______    SETZE  "W  :W+1 ___________________________________
 _______    BESTUECKE  "A  :R  ( KOMPONENTE  "A  :R ) + 1 ]
 RUECKGABE  :W
ENDE

PR KEIN.VOLLSTAENDIGER.SATZ?
 RUECKGABE ( :A.1 * :A.2 * :A.3 * :A.4 * :A.5 * :A.6 )  = 0
ENDE
```

Beispiele:

DEF.FELD "A 6
KEIN.VOLLSTAENDIGER.SATZ?
ERGEBNIS: WAHR

FUER [I 1 6 [BESTUECKE "A :I :I]]
KEIN.VOLLSTAENDIGER.SATZ?
ERGEBNIS: FALSCH

WARTEZEIT
ERGEBNIS: 18

WH 100 [DZ WARTEZEIT]
15
12
...
17

Aufgabe 7.6: Bauen Sie das Simulationsbeispiel aus!
(a) Betten Sie die Funktion WARTEZEIT in eine Testserie ein.
(b) Ermitteln Sie die minimale, die maximale, die durchschnittliche und die häufigste Wartezeit.
(c) Legen Sie eine Tabelle für die Häufigkeiten der einzelnen Wartezeiten an ("Häufigkeitsverteilung").
(d) Ein kleines Projekt: Schreiben Sie einen Modul HISTOGRAMM, mit dem Sie Tabellen in Stäbchendiagramme umsetzen können. Stellen Sie die Häufigkeitsverteilung aus (c) graphisch dar.

Aufgabe 7.7: Simulieren Sie das Roulette-Spiel in Logo.

Aufgabe 7.8: Simulieren Sie das folgende "Glücksrad" in Logo.

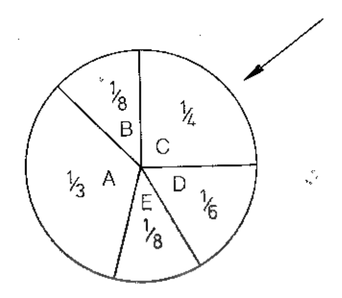


Abbildung 7.3: Glücksrad

Aufgabe 7.9: Modellieren Sie weitere Zufallsprozesse im Sinne des obigen Beispiels.

Kapitel 8: Programme als Daten; Daten als Programme

8.1 PRLISTE legt die interne Struktur von Prozeduren offen

In Kapitel 3 haben wir gesehen, wie man mit Hilfe des Logo-Editors Prozeduren schreiben kann. Die Prozedur ist dabei zunächst nichts anderes als ein beliebiger Text. Mit ähnlichen Texteditoren erstellt man zum Beispiel Briefe, Artikel oder ganze Bücher.

Wenn wir mit dem "Lern"-befehl Ctrl-C aus dem Logo-Editor aussteigen, wird die Prozedur (automatisch) in eine ausführbare Form gebracht: Der im Editor vorliegende Prozedurtext wird dabei in eine **Prozedurliste** umgewandelt.

Zwischen Prozedurlisten und den bisher behandelten "gewöhnlichen" (Daten-) Listen gibt es aus syntaktischer Sicht überhaupt keinen Unterschied. Die Logo-Grundfunktion PRLISTE :P (kurz PL :P) gibt als Funktionswert gerade die Prozedurliste der Prozedur mit Namen :P aus. (Unglücklicherweise heißt diese Funktion in der englischsprachigen MIT-Version von Logo TEXT, obwohl sie gerade nicht den Prozedurtext sondern die Prozedurliste als Ausgabe hat).

Als Beispiel betrachten wir die Prozedur POLYGON :N :S, mit der wir (regelmäßige) Polygone der Eckenzahl :N und der Seitenlänge :S zeichnen können:

```
PR POLYGON  :N  :S
 WH   :N [ VW  :S  RE 360 / :N ]
ENDE
```

Der Aufruf POLYGON 5 60 liefert zum Beispiel das folgende Fünfeck:

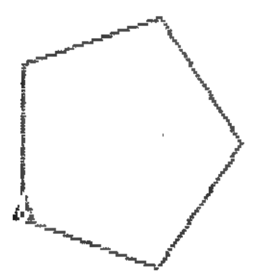


Abbildung 8.1: POLYGON 5 60

Die Prozedurliste von POLYGON erhalten wir durch den folgenden Aufruf:

PRLISTE "POLYGON
ERGEBNIS: [[:N :S] [WH :N [VW :S RE 360 / :N]]]

Die Prozedurliste wird allgemein nach den folgenden Gesetzmäßigkeiten gebildet:

(1.) Die erste Teilliste der Prozedurliste enthält genau die formalen Eingabeparameter der Prozedur. Hat die Prozedur keine Eingabeparameter, so bleibt die erste Teilliste leer. Beispiel:

```
PR KASTEN
 WH  2 [ VW 50   RE 90   VW 100   RE 90 ]
ENDE
```

PRLISTE "KASTEN
ERGEBNIS: [[] [WH 2 [VW 50 RE 90 VW 100 RE 90]]]

(2.) Jede logische Zeile des Prozedurtextes wird in eine Teilliste der Prozedurliste umgewandelt. Enthält die Zeile ihrerseits eine oder mehrere, möglicherweise auch geschachtelte Listen, so werden sie in genau derselben Schachtelungsstruktur in die Zeilen-Teilliste übernommen. Beispiel:

```
PR DREI.KAESTEN.UEBEREINANDER  :B  :L
 WH 3  [ WH 2  [ VW :B  RE 90  VW :L  RE 90 ]  VW :B ]
ENDE
```

(Der Eingabeparameter :B steht dabei für Breite, :L für Länge).

PRLISTE "DREI.KAESTEN.UEBEREINANDER
ERGEBNIS: [[:B :L] [WH 3 [WH 2 [VW :B RE 90 VW :L RE 90] VW :B]]]

(3.) Die definierenden Wörter PR, ENDE und der in der Definition auftretende Prozedurname kommen in der Prozedurliste nicht vor.

Auf die Prozedurliste können nun alle Listenoperationen angewandt werden, zum Beispiel:

ERSTES PRLISTE "KASTEN
ERGEBNIS: [ ]

LETZTES LETZTES LETZTES PRLISTE "KASTEN
ERGEBNIS: 90

Auch der TUE-Befehl kann im Prinzip auf die Prozedurlisten angewandt werden. An Stelle von KASTEN könnte man auch stets die Kommandofolge TUE LETZTES PRLISTE "KASTEN aufrufen.

Aufgabe 8.1: Es sei :L die Prozedurliste von DREI.KAESTEN.UEBEREINANDER.
(a) Schreiben Sie Prozeduren, mit denen Sie jede der in :L vorkommenden Teillisten herausfiltern können.
(b) Schreiben Sie eine Prozedur, mit der Sie alle in :L vorkommenden Wörter (bzw. Zahlen) "herausfischen" können.

8.2 DEF ermöglicht die Umwandlung einer Liste in eine Prozedur

Die folgende Prozedur zeichnet einen zunehmenden Mond:

```
PR  ZUNEHMENDER.MOND
 BILD  VERSTECKIGEL
 RECHTS 90
 WIEDERHOLE 60  [ VORWAERTS 2  LINKS 3 ]
 STIFTHOCH  MITTE  STIFTAB
 RECHTS 120
 WIEDERHOLE 25  [ VORWAERTS 8  LINKS 10 ]
ENDE
```



Abbildung 8.2: zunehmender Mond

Die Prozedurliste des zunehmenden Mondes erhalten wir durch:

PRLISTE "ZUNEHMENDER.MOND

```
ERGEBNIS:  [ [ ] [ BILD VERSTECKIGEL ] ___________
             [RECHTS 90 ] _____________________
             [WIEDERHOLE 60 [VORWAERTS 2 LINKS 3 ] ] ____
             [STIFTHOCH MITTE STIFTAB ] ____________
             [RECHTS 120] ______________________
             [ WIEDERHOLE 25 [ VORWAERTS 8 LINKS 10 ] ] ]
```

Wir wollen uns jetzt der Frage zuwenden, was man mit diesen Prozedurlisten tun kann. Stellen Sie sich vor, wir möchten einen abnehmenden Mond zeichnen. Eine Möglichkeit zur Lösung dieses Problems besteht darin, daß wir die Prozedur ZUNEHMENDER.MOND reeditieren und alle RECHTS-Befehle durch LINKS-Befehle beziehungsweise alle LINKS-Befehle durch RECHTS-Befehle ersetzen.

Andererseits haben wir in Kapitel 5, Abschnitt 5.6, eine Prozedur ERSETZE :ALT :NEU :L kennengelernt, mit deren Hilfe wir in der Liste :L das durch :ALT gegebene Wort durch :NEU ersetzen können, wo immer es (in :L) vorkommt. Lassen Sie uns etwas mit der Prozedur ERSETZE experimentieren:

```
ERSETZE "LINKS "RECHTS ( PRLISTE "ZUNEHMENDER.MOND )
ERGEBNIS:  [ [ ] [BILD VERSTECKIGEL ] ___________
             [RECHTS 90 ] _____________________
             [WIEDERHOLE 60 [VORWAERTS 2 RECHTS 3 ] ] ___
             [STIFTHOCH MITTE STIFTAB ] ____________
             [RECHTS 120] ______________________
             [ WIEDERHOLE 25 [ VORWAERTS 8 RECHTS 10 ] ] ]
```

Wir haben die Prozedurliste zwar etwas manipuliert aber sicher noch nicht das gewünschte Ziel erreicht. Dazu müßten wir natürlich parallel auch alle RECHTS-Befehle in LINKS-Befehle umwandeln. Es liegt hier eigentlich ein kleines Übersetzungsproblem vor.

Zunächst einmal schaffen wir uns eine Prozedur, die uns ein altes Wort durch ein neues Wort ersetzt:

```
PR ERSETZE.WORT  :ALT
 WENN  :ALT  =  "RECHTS  RUECKGABE  "LINKS
 WENN  :ALT  =  "LINKS  RUECKGABE  "RECHTS
 RUECKGABE  :ALT
ENDE
```

Ein Wort, das nicht ausgetauscht werden soll, wird unverändert zurückgegeben. Hier einige Beispiele:

ERSETZE.WORT "RECHTS
ERGEBNIS: LINKS

ERSETZE.WORT "XAVER
ERGEBNIS: XAVER

Mit Hilfe der ERSETZE.WORT-Funktion können wir jetzt eine Übersetzungs-Prozedur schreiben, welche in (Prozedur-) Listen die in ERSETZE.WORT spezifizierten Wörter durch ihre neuen Versionen ersetzt.

```
PR UEBERSETZUNG :L
 WENN :L = [ ]  DANN  RUECKGABE :L
 WENN LISTE? ERSTES :L  DANN RUECKGABE
             MITERSTEM ( UEBERSETZUNG ERSTES :L )
                       ( UEBERSETZUNG OHNEERSTES :L )
 RUECKGABE MITERSTEM ( ERSETZE.WORT ERSTES :L )
                     ( UEBERSETZUNG OHNEERSTES :L )
ENDE
```

Beachten Sie die strukturelle Ähnlichkeit zwischen den Prozeduren ERSETZE (aus Kapitel 5) und UEBERSETZUNG.

Ein Beispiel:

UEBERSETZUNG PRLISTE "ZUNEHMENDER.MOND

```
ERGEBNIS:   [ [ ] [ BILD VERSTECKIGEL ]
              [LINKS 90]
              [WIEDERHOLE 60 [ VORWAERTS 2 RECHTS 3 ] ]
              [STIFTHOCH MITTE STIFTAB ]
              [LINKS 120 ]
              [ WIEDERHOLE 25 [ VORWAERTS 8 RECHTS 10 ] ] ]
```

Mit dem Befehl DEF :N :L wird nun die Prozedur mit der Prozedurliste :L unter dem Namen :N definiert. Ein Beispiel:

DEF "HOCH.DREI [[:X] [RUECKGABE :X * :X * :X]]

Die so definierte Prozedur HOCH.DREI kann wie jede andere Prozedur gehandhabt werden.

HOCH.DREI 5
ERGEBNIS: 125

EDIT HOCH.DREI (im LCSI Logo: EDIT "HOCH.DREI)
Logo schaltet in den Editiermodus um und bringt:

```
PR HOCH.DREI :X
 RUECKGABE :X * :X * :X
ENDE
```

Bei der Ausführung von DEF wird nicht geprüft, ob der Inhalt der Liste :L syntaktisch einer Prozedur entspricht. Wenn man will, kann man auch folgendes tun:

DEF "KOMISCHE.PROZEDUR [[ALPHA BETA] [GAMMA DELTA EPSILON]]
EDIT KOMISCHE.PROZEDUR

```
PR KOMISCHE.PROZEDUR  ALPHA BETA
 GAMMA DELTA EPSILON
ENDE
```

(Aus dieser "Prozedur" sollte man mit Ctrl-G aussteigen und sie so schnell wie möglich vergessen).

Zurück zu unseren Monden. Durch den Aufruf

DEF "ABNEHMENDER.MOND UEBERSETZUNG PRLISTE "ZUNEHMENDER.MOND

definieren wir die neue Prozedur ABNEHMENDER.MOND, ohne sie zu editieren. Der Aufruf ABNEHMENDER.MOND ergibt nun schließlich das folgende Bild:



Abbildung 8.3: abnehmender Mond

Natürlich können wir auch das DEF-Kommando noch in eine Prozedur verpacken:

```
PR ACHSENSPIEGELUNG  :PN
 DEF ( WORT "GESPIEGELT.  :PN )  UEBERSETZUNG  PRLISTE  :PN
ENDE
```

Die Prozedur ACHSENSPIEGELUNG führt insgesamt folgendes aus: Die durch den Namen :PN (für Prozedur-Name) gegebene Prozedur wird übersetzt. An den alten Namen wird vorn das Wort GESPIEGELT. angehängt, und das Übersetzungsergebnis wird unter diesem erweiterten Namen als Prozedur definiert.

Ein Beispiel:

ACHSENSPIEGELUNG "ZUNEHMENDER.MOND

Hier ein Test, ob wir wirklich das Gewünschte erreicht haben:

(PRLISTE "GESPIEGELT.ZUNEHMENDER.MOND) = (PRLISTE "ABNEHMENDER.MOND)
ERGEBNIS: WAHR

Die Prozeduren GESPIEGELT.ZUNEHMENDER.MOND und ABNEHMENDER.MOND sind also gleich (bis auf ihren Namen).

Aufgabe 8.2: Schreiben Sie eine Prozedur PUNKTSPIEGELUNG als Analogon zur Prozedur ACHSENSPIEGELUNG.

Aufgabe 8.3: Schreiben Sie eine Prozedur HALBMOND, mit der Sie Halbmonde in verschiedenen Größen zeichnen können. Überlegen Sie, was für Eingabeparameter die Prozedur HALBMOND haben sollte. Wenden Sie die Übersetzungsprozedur auf die Prozedurliste von HALBMOND an.

Aufgabe 8.4: Schreiben Sie eine Prozedur DEFUEB :NEU :ALT, mit der Sie die Ausgangsprozedur :ALT übersetzen und unter dem Namen :NEU definieren können. Die Prozedur DEFUEB sollte folgendermaßen aufrufbar sein:
DEFUEB "ABNEHMENDER.MOND "ZUNEHMENDER.MOND

Nun wollen wir uns noch einem schwierigeren Beispiel zuwenden. Nehmen wir an, wir haben eine Prozedur BAUM.IM.OSTWIND geschrieben:

```
PR BAUM.IM.OSTWIND  :GROESSE
 WENN  :GROESSE  < 2 DANN RUECKKEHR
 VORWAERTS  :GROESSE
 LINKS 60
 BAUM.IM.OSTWIND  :GROESSE * 0.5
 RECHTS 55
 BAUM.IM.OSTWIND  :GROESSE * 0.8
 RECHTS 5
 RUECKWAERTS  :GROESSE
ENDE
```

So sieht der Baum im Ostwind aus (mit :GROESSE = 40, y-Koordinate des Startpunkts bei etwa -60):

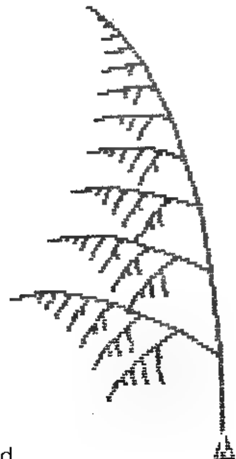


Abbildung 8.4: Baum im Ostwind

Wir wollen aber auch einen Baum im Westwind zeichnen können. Lassen Sie uns zunächst einmal ganz schematisch wie beim Beispiel des zunehmenden Mondes verfahren.

ACHSENSPIEGELUNG "BAUM.IM.OSTWIND

Dies führt zur Definition der Prozedur GESPIEGELT.BAUM.IM.OSTWIND. Wenn wir diese Prozedur ausführen, so erhalten wir das folgende merkwürdige Bild:

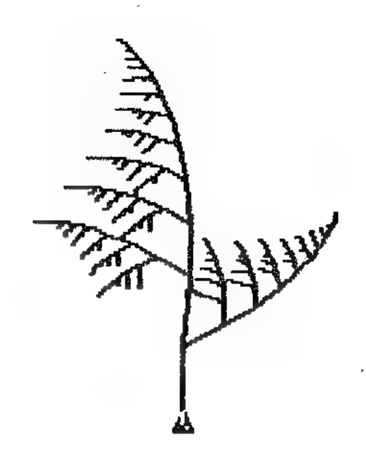


Abbildung 8.5: GESPIEGELT.BAUM.IM.OSTWIND 40

Da wir offensichtlich nicht das richtige Ergebnis erhalten haben, schauen wir uns jetzt die Prozedur GESPIEGELT.BAUM.IM.OSTWIND an:

EDIT GESPIEGELT.BAUM.IM.OSTWIND

```
PR GESPIEGELT.BAUM.IM.OSTWIND :GROESSE
 WENN :GROESSE < 2 DANN RUECKKEHR
 VORWAERTS :GROESSE
 RECHTS 60
 BAUM.IM.OSTWIND :GROESSE * 0.5
 LINKS 55
 BAUM.IM.OSTWIND :GROESSE * 0.8
 LINKS 5
 RUECKWAERTS :GROESSE
ENDE
```

Damit ist klar, warum die Prozedur nicht zum gewünschten Ergebnis geführt hat. BAUM.IM.OSTWIND ist rekursiv und ruft sich selbst auf. Entsprechend müßte sich auch die Prozedur GESPIEGELT.BAUM.IM.OSTWIND selbst aufrufen. Stattdessen geht sie "fremd" und ruft BAUM.IM.OSTWIND auf.

Das Problem läßt sich auf (mindestens) zwei Arten beheben. Zum einen können wir den Fehler einfach im Editier-Modus "von Hand" korrigieren.

Es wäre jedoch schöner, wenn die Ausgangsprozedur BAUM.IM.OSTWIND gleich richtig übersetzt würde. Dies ist nicht schwer. Wir müssen einfach den Begriff BAUM.IM.OSTWIND als zu übersetzendes Wort in die Prozedur ERSETZE.WORT aufnehmen. Die Übersetzung von BAUM.IM.OSTWIND sei diesmal BAUM.IM.WESTWIND.

```
PR ERSETZE.WORT  :ALT
 WENN  :ALT  =  "RECHTS  RUECKGABE  "LINKS
 WENN  :ALT  =  "LINKS   RUECKGABE  "RECHTS
 WENN  :ALT  =  "BAUM.IM.OSTWIND  RUECKGABE  "BAUM.IM.WESTWIND
 RUECKGABE  :ALT
ENDE
```

Die Übersetzungs-Prozedur braucht nicht verändert zu werden. Führen wir den Übersetzungsprozeß jetzt zu Ende.

DEF "BAUM.IM.WESTWIND UEBERSETZUNG PRLISTE "BAUM.IM.OSTWIND

BAUM.IM.WESTWIND 20



Abbildung 8.6: Baum im Westwind

Das ist es, was wir gewollt haben. Überzeugen Sie sich auch durch ZEIGE BAUM.IM.WESTWIND. Zum Abschluß noch das folgende hübsche Bild:

BAUM.IM.OSTWIND 40 BAUM.IM.WESTWIND 40

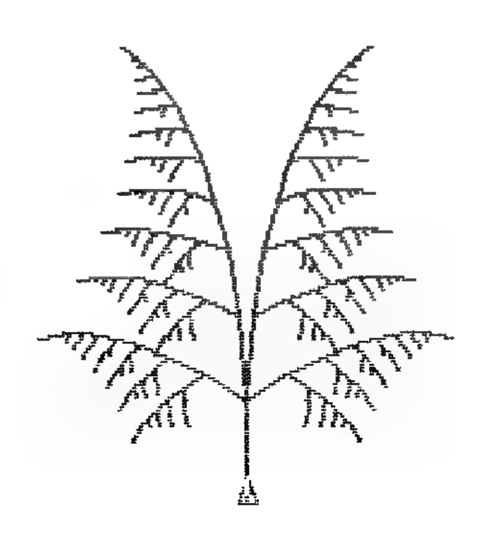


Abbildung 8.7: Ostwind - Westwind

8.3 UEBERSETZUNG: ein Programm übersetzt Programme - auch sich selbst

Die Prozedur UEBERSETZUNG aus dem vorangehenden Abschnitt ist universeller als es auf den ersten Blick scheint. Mit ihr können wir nicht nur RECHTS durch LINKS und LINKS durch RECHTS und BAUM.IM.OSTWIND durch BAUM.IM.WEST-WIND ersetzen, sondern wir können jede Sprache in jede strukturgleiche Sprache übersetzen. Die einzige Vorausetzung hierzu ist, daß wir ein entsprechendes Wörterbuch zur Verfügung haben.

Das in Abschnitt 8.2 benutzte Wörterbuch war ziemlich mager. Es bestand nur aus drei Wörtern und ihren Entsprechnungen:

Quellensprache	Zielsprache
LINKS	RECHTS
RECHTS	LINKS
BAUM.IM.OSTWIND	BAUM.IM.WESTWIND

Um englischsprachige Logo-Prozeduren in deutschsprachige Prozeduren zu übersetzen, müssen wir als Wörter der Quellensprache nur die englischsprachigen Logo-Grundwörter und als Wörter der Zielsprache die dazugehörigen Grundwörter der deutschen Logo-Version verwenden. Das Wörterbuch sieht in diesem Fall folgendermaßen aus:

Logo-Grundwörter in	
Logo-Englisch	Logo-Deutsch
ALL	ALLES
ALLOF	ALLE?
ANYOF	EINES?
...	...
BF	OE
BUTFIRST	OHNEERSTES
...	...
WRAP	RANDSPRUNG
XCOR	XKO
YCOR	YKO

Das Wörterbuch war bisher in der Prozedur ERSETZE.WORT verankert. Man könnte mit diesem neuen Wörterbuch im Prinzip genauso verfahren:

```
PR ERSETZE.WORT  :X
 WENN  :X  =  "ALL    RUECKGABE  "ALLES
 WENN  :X  =  "ALLOF  RUECKGABE  "ALLE?
 ...
 WENN  :X  =  "YCOR   RUECKGABE  "YKO
 RUECKGABE  :X
ENDE
```

Aufgabe 8.5: Geben Sie Beispiele dafür an, wo das Wort :X unverändert zurückgegeben wird.

Wie in der erweiterten Version von ERSETZE.WORT im vorigen Abschnitt kann man natürlich auch Namen selbstgeschriebener Prozeduren in das Wörterbuch aufnehmen.

Es gibt aber noch eine andere, vorteilhaftere Möglichkeit, um das Wörterbuch zu verankern. Erinnern Sie sich an die Methode aus Kapitel 4, Abschnitt 4.2? Im Sinne dieser Methode wären die englischsprachigen Grundwörter als Variablennamen zu deuten, deren Wert das jeweilige deutsche Grundwort ist. Also etwa folgendermaßen:

```
SETZE  "ALL    "ALLES
SETZE  "ALLOF  "ALLE?
SETZE  "YCOR   "YKO
```

Natürlich müssen wir zur Benutzung dieses Wörterbuchs eine andere Version von ERSETZE.WORT schreiben. Hierfür erweist sich die prädikative Prozedur NAME? als äußerst nützlich. Mit ihr können wir feststellen, ob ein bestimmtes Wort ein (Variablen-) Name ist oder nicht:

```
PR ERSETZE.WORT  :X
 WENN  NAME?  :X  DANN  RUECKGABE  WERT  :X
 RUECKGABE  :X
ENDE
```

Einige Beispiele:

```
ERSETZE.WORT "BUTFIRST
ERGEBNIS: OHNEERSTES

ERSETZE.WORT "ISODOR
ERGEBNIS: ISODOR

ERSETZE.WORT "+
ERGEBNIS: +

ERSETZE.WORT "(
ERGEBNIS: (
```

Im Prinzip haben wir damit das Problem der Übersetzung englischsprachiger Prozeduren des MIT Logo in deutschsprachige Prozeduren gelöst. Nehmen wir zum Beispiel die englischsprachige Prozedur

```
TO SIGN  :X
 IF  :X  >  0  THEN  OUTPUT  1
 IF  :X  <  0  THEN  OUTPUT  (-1)
 OUTPUT  0
END
```

Mit unserer neuen ERSETZE.WORT - Prozedur liefern die folgenden Kommandos

```
DEF "SIGNUM UEBERSETZUNG PRLISTE "SIGN
EDIT SIGNUM
```

das Ergebnis:

```
PR SIGNUM :X
 WENN :X > 0 DANN RUECKGABE 1
 WENN :X < 0 DANN RUECKGABE (-1)
 RUECKGABE 0
ENDE
```

Wenn die englischsprachige Prozedur auch SIGNUM geheißen hätte, wäre sie zwar richtig übersetzt worden, aber da für die übersetzte Version derselbe Name wie für die Ausgangsversion verwendet worden wäre, stünde sie nicht mehr zur Verfügung. Sie wäre durch die neue (deutsche) Version überschrieben worden - so, wie es stets der Fall ist, wenn wir einen schon vorhandenen Prozedurnamen bei der Definition einer neuen Prozedur verwenden.

Wieso ist eigentlich das TO in PR und das END in ENDE "übersetzt" worden, obwohl diese Wörter doch gar nicht in der Porzedurliste von SIGNUM auftreten? Nun: TO, PR, END und ENDE gehören nicht direkt zur Prozedurliste; sie werden einfach beim Editieren durch das Logo-System hinzugefügt. In der deutschen Version PR und ENDE; in der englischen Version TO und END.

Das Einlesen von Prozeduren von der Diskette geschieht ganz analog zum Eintippen. Der Disketteninhalt wird zunächst in den Editierspeicher geschrieben und dann gelernt, als ob jemand Ctrl-C eingegeben hätte.

Dabei ist die deutsche Logo-Version noch eine Spur "intelligenter" als die englische, denn sie versteht auch die für das "Erlernen" der Prozeduren entscheidenden Grundwörter TO, END und MAKE. Sie haben im deutschen Logo dieselbe Wirkung wie PR, ENDE und SETZE. Probieren wir es in der deutschen Logo-Version für den Apple II Computer einmal aus:

TO TEST

```
PR TEST
 DRUCKEZEILE [ ES KLAPPT TATSAECHLICH ]
END
Ctrl-C
```

Logo reagiert mit: TEST GELERNT .

TEST
ES KLAPPT TATSAECHLICH

Die mit TO definierte Prozedur TEST hat jetzt genau denselben Status wie jede andere selbstgeschriebene Prozedur und läßt sich auch ganz normal editieren und abspeichern. Auch Prozeduren, deren Definition mit PR eingeleitet wurde, können im Prinzip mit Hilfe von TO reeditiert werden.

An dieser Stelle ist noch ein kleiner Hinweis zur Laufzeiteffizienz von Logo angebracht. Logo-Prozeduren laufen meist in der Tat erheblich langsamer als entsprechende Prozeduren in anderen Programmiersprachen; besonders dann, wenn man compilierende Sprachen zum Vergleich heranzieht. Andererseits erlaubt das Sprachkonzept von Logo gelegentlich Lösungen, die sowohl von der Klarheit der

Formulierung als auch von der Laufzeiteffizienz anderen Sprachen überlegen sind. So dürfte zum Beispiel die neue Version der Prozedur ERSETZE.WORT auch in der Laufzeiteffizienz von keiner anderen höheren Programmiersprache zu übertreffen sein. Denn aufgrund des Variablenkonzepts von Logo entfällt der Suchprozeß, wie er etwa in der alten Version ERSETZE.WORT für ein verschwindend kleines Vokabular ausgeführt ist, völlig. Da in kaum einer anderen Sprache außer Lisp/Logo die Unterscheidung zwischen dem Namen und dem Wert von Variablen praktiziert wird, sind derartige Lösungen dort gar nicht möglich.

Es darf aber auch nicht übersehen werden, daß Laufzeiteffizienz und Speicherplatzeffizienz nur zwei **Effizienz-Aspekte** sind. Häufig bleibt bei der Diskussion um die Frage der Effizienz von Programmiersprachen ein Aspekt unerwähnt, nämlich: wie lange dauert es, um ein Programm für eine bestimmte Problemlösung zu schreiben, wie "natürlich" ist die Problemlösung, wie gut spiegelt das entsprechende Programm das ursprüngliche Problem wider? Diese letzteren Effizienzfragen werden gelegentlich auch als Probleme der kognitiven Effizienz bezeichnet. In diesem Bereich liegen die besonderen Stärken der Lisp/Logo-Sprachfamilie.

Abschließend noch einige praktische Hinweise zur "Implementierung" des Wörterbuches. So wie wir es oben getan haben, hat es den Nachteil, daß die Wörterbuch-Variablen mit anderen globalen Variablen vermischt werden. Dies macht zwar der Prozedur ERSETZE.WORT keine Schwierigkeiten, es erschwert aber den Überblick über das Wörterbuch für den Programmierer. Besonders dann, wenn man das Wörterbuch im Logo-Editor einmal abändern möchte, was im Grunde ja möglich ist, da man auch die Variablen editieren kann, erweist sich die fehlende Bündelung der im Wörterbuch vorkommenden Namen als nachteilig.

Ein kleiner Trick verschafft hier Abhilfe. Wir schreiben eine Scheinprozedur WOERTERBUCH, in deren Prozedurliste jeweils Originalwort und Ersetzungswort als ein Listenpaar auftreten:

```
PR WOERTERBUCH
  .ASPECT  .SKALA
  ...
  ALL  ALLES
  ...
  FPUT  MITERSTEM
  ...
  QUOTIENT  DIV
  ...
  STOP RUECKKEHR
  ...
  THING  WERT
  ...
  YCOR  YKO
ENDE
```

Ein vollständiges Verzeichnis der englischen und deutschen Grundwörter befindet sich im Anhang. Diese Scheinprozedur WOERTERBUCH soll natürlich nicht laufen. Wir benötigen nur ihre Prozedurliste:

```
[ [ ] ____________________________________________
_______ [ .ASPECT .SKALA ] ________________________
_______ ... _______________________________________
_______ [ ALL ALLES ] _____________________________
_______ ... _______________________________________
_______ [ FPUT MITERSTEM ] ________________________
_______ ... _______________________________________
_______ [ YCOR YKO ] ]
```

Mit der folgenden Prozedur erstellen wir nun aus den Begriffen der Scheinprozedur das Wörterbuch; der Abwechslung halber einmal zum Zwecke der Übersetzung von Logo-Deutsch nach Logo-Englisch.

```
PR MACHE.WOERTERBUCH
 MACHE.WOERTERBUCH.1  OHNEERSTES  PRLISTE  "WOERTERBUCH
ENDE

PR MACHE.WOERTERBUCH.1  :L
 WENN  :L  =  [ ]  DANN  RUECKKEHR
 SETZE  LETZTES  ERSTES  :L  ERSTES  ERSTES  :L
 MACHE.WOERTERBUCH.1  OHNEERSTES  :L
ENDE
```

Aufgabe 8.6: Schreiben Sie eine Prozedur MACHE.WOERTERBUCH.ENGLISCH-.DEUTSCH, welche die deutschen Begriffe des Wörterbuches als Werte der englischen Begriffe definiert. (Das Wörterbuch selbst soll dabei natürlich nicht noch einmal von Hand eingetippt werden).

Nachdem die Prozedur MACHE.WOERTERBUCH ausgeführt wurde, laufen die Prozeduren ERSETZE.WORT und UEBERSETZUNG unverändert.

Die Prozedur UEBERSETZUNG kann schließlich auch auf sich selbst angewandt werden. Dabei ist es sinnvoll, die Namen der zu übersetzenden Prozeduren in das Wörterbuch aufzunehmen:

```
SETZE "UEBERSETZUNG "TRANSLATION
SETZE "ERSETZE.WORT "REPLACE.WORD
DEF "TRANSLATION UEBERSETZUNG PRLISTE "UEBERSETZUNG
EDIT TRANSLATION
```

```
PR TRANSLATION  :L
 IF  :L = [ ]  THEN  OUTPUT  :L
 IF  LIST?  FIRST  :L  THEN  OUTPUT ____________________
 _____________ FPUT ( TRANSLATION  FIRST  :L ) _________
 _____________      ( TRANSLATION  BUTFIRST  :L )
 OUTPUT  FPUT ( REPLACE.WORD  FIRST  :L ) ______________
 __________   ( TRANSLATION  BUTFIRST  :L )
ENDE
```

Aufgabe 8.7: Übersetzen Sie noch die Prozedur ERSETZE.WORT, um so in Verbindung mit Aufgabe 8.6 eine komplette Übersetzungs-Umgebung von Logo-Englisch nach Logo-Deutsch zu bekommen.

8.4 HILBERT: eine Prozedur die sich selbst verändert, während sie läuft

Intuitiv scheint klar zu sein, was eine Kurve und was eine Fläche ist. In Abbildung 8.8 sind Vertreter der jeweiligen Spezies skizziert.

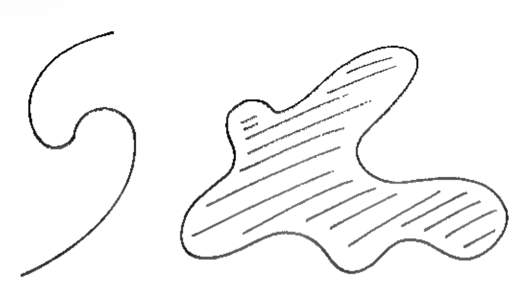


Abbildung 8.8: Kurve und Fläche

Unter einer stetigen Kurve wollen wir hier anschaulich eine Kurve verstehen, die man mit dem Finger ohne abzusetzen "nachfahren" kann.

Die Vorstellung, daß eine stetige Kurve eine vorgegebene Fläche lückenlos ausfüllen kann, erscheint zunächst abwegig. Dennoch gibt es solche Kurven. Eine der bekanntesten wurde von David Hilbert (deutscher Mathematiker 1862 - 1943) entdeckt. Die HILBERT-Prozedur, welche wir im folgenden betrachten werden, zeichnet in Abhängigkeit vom Wert des Parameters STUFE eine kreuzungsfreie Schlängellinie innerhalb eines bestimmten Quadrates. Diese Schlängellinie ist ein Beispiel für eine stetige Kurve.

Die folgende Version der Hilbert-Kurve findet man im wesentlichen in dem grundlegenden Buch "Turtle Geometry" von H.Abelson und A. diSessa, MIT Press, Cambridge, Massachusetts, 1980.

```
PR HILBERT  :GROESSE  :STUFE  :PARITAET
  WENN  :STUFE  =  0   DANN  RUECKKEHR
  LINKS  :PARITAET * 90
  HILBERT  :GROESSE  ( :STUFE - 1 )  ( - :PARITAET )
  VORWAERTS  :GROESSE
  RECHTS  :PARITAET * 90
  HILBERT  :GROESSE  ( :STUFE - 1 )  :PARITAET
  VORWAERTS  :GROESSE
  HILBERT  :GROESSE  ( :STUFE - 1 )  :PARITAET
  RECHTS  :PARITAET * 90
  VORWAERTS  :GROESSE
  HILBERT  :GROESSE  ( :STUFE - 1 )  ( - :PARITAET )
  LINKS  :PARITAET *  90
ENDE
```

Nehmen wir jetzt an, daß wir verschiedene Hilbert-Kurven in ein Quadrat mit festen äußeren Abmessungen zeichnen. Je größer also die Stufenzahl ist, desto kleiner

muß :GROESSE gewählt werden. Je größer wir :STUFE wählen, desto besser füllt die Kurve das Quadrat aus. Wenn :STUFE gegen unendlich geht, füllt die Grenzwertkurve das Quadrat lückenlos aus.

Um die HILBERT-Prozedur zu verstehen, ist es hilfreich, einige Versionen mit kleiner Stufenzahl zu zeichnen. Es kann auch nützlich sein, den Lauf von HILBERT im Protokoll-Modus zu verfolgen. Vor dem Aufruf der nächsten drei Hilbert-Kurven wurde jeweils BILD und VI eingegeben. Bei der Stufenzahl 0 wird gar nichts gezeichnet. Die Stufenzahl 1 liefert die Grundfigur in der Form eines eckigen, liegenden U.

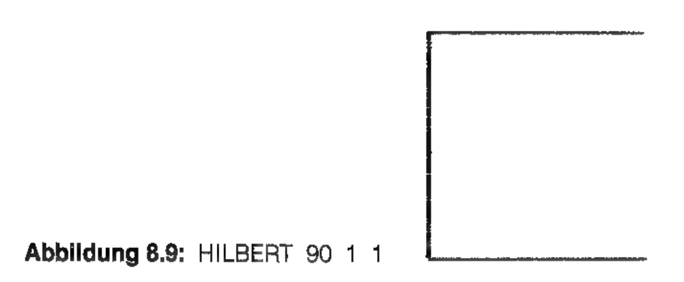


Abbildung 8.9: HILBERT 90 1 1

Hier noch die nächst höheren Stufenzahlen:

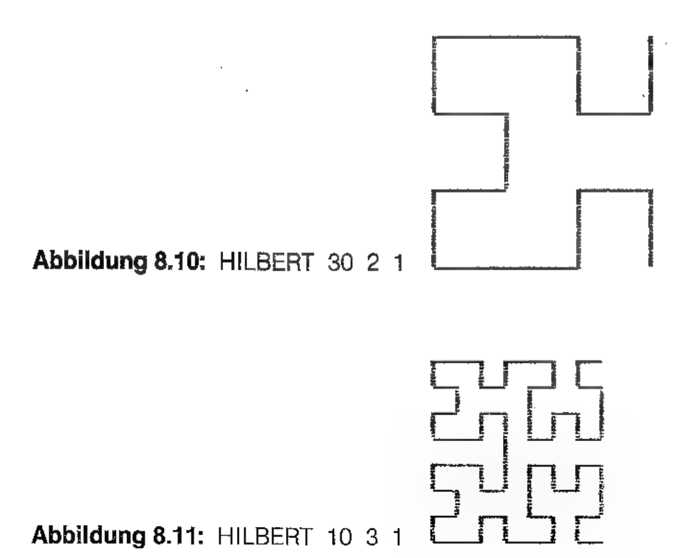


Abbildung 8.10: HILBERT 30 2 1

Abbildung 8.11: HILBERT 10 3 1

Der Übergang von der Stufenzahl 1 zur Stufenzahl 2 macht deutlich, was sich in der HILBERT - Prozedur abspielt. HILBERT 30 2 1 entsteht aus HILBERT 90 1 1 dadurch, daß wir letztere Figur durch vier verkleinerte Kopien ersetzen, die noch durch geeignete Anschlußstücke zu verbinden sind.

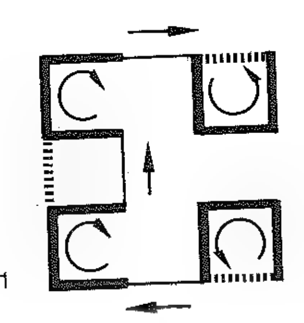


Abbildung 8.12: HILBERT 30 2 1 eingebettet in HILBERT 90 1 1

Allerdings werden die vier kleinen Hilbert-Grundformen nicht alle in derselben Orientierung durchlaufen. Die Orientierung der ersten und vierten Grundfigur ist gerade umgekehrt wie in der Ausgangsfigur (Stufe 1); die zweite und dritte Grundfigur haben dieselbe Orientierung wie die Figur in der Stufe 1.

Dafür, daß die kleinen Grundfiguren jeweils in der richtigen Orientierung durchlaufen werden, sorgt der Parameter, der als Parität bezeichnet wurde. Er schwankt zwischen 1 und -1 hin und her und verursacht dadurch jeweils den gewünschten Orientierungswechsel.

Die Rolle der Parität ist manchmal nicht ganz leicht zu verstehen. Wir wollen deshalb zum Zwecke der Verdeutlichung einen anderen Weg gehen.

Ein Orientierungswechel würde sicher auch dadurch erreicht, daß wir in der Prozedur HILBERT alle LINKS-Befehle durch RECHTS-Befehle und alle RECHTS-Befehle durch LINKS-Befehle ersetzen. Dies ist im Prinzip kein Problem. Wir haben diese Methode zum Beispiel schon beim Übergang von der Prozedur BAUM.IM.OSTWIND zur Prozedur BAUM.IM.WESTWIND angewandt.

Hier liegt jedoch eine neue Situation vor: die Prozedur HILBERT muß sich selbst ändern, während sie läuft und dann in der geänderten Form weiterlaufen.

Daß auch dies möglich ist, zeigt die folgende Version von HILBERT, zu der noch die Hilfsprozedur RICHTUNG.AENDERN gehört. Da RICHTUNG.AENDERN auf UEBERSETZUNG zurückgreift, muß zu Beginn, zum Beispiel mit Hilfe der Prozedur MACHE.WOERTERBUCH.HILBERT.KURVE, das entsprechende Wörterbuch definiert werden. Damit es keine unliebsamen Störungen gibt, sollten vorher alle globalen Variablen gelöscht werden.

```
PR HILBERT.ANDERS  :GROESSE  :STUFE
 WENN  :STUFE = 0  DANN  RUECKKEHR
 LINKS  90
 RICHTUNG.AENDERN
 HILBERT.ANDERS  :GROESSE  ( :STUFE - 1 )
 RICHTUNG.AENDERN
 VORWAERTS  :GROESSE
 RECHTS  90
 HILBERT.ANDERS  :GROESSE  ( :STUFE - 1 )
 VORWAERTS  :GROESSE
 HILBERT.ANDERS  :GROESSE  ( :STUFE - 1 )
```

```
 RECHTS 90
 VORWAERTS :GROESSE
 RICHTUNG.AENDERN
 HILBERT.ANDERS :GROESSE ( :STUFE - 1 )
 RICHTUNG.AENDERN
 LINKS 90
ENDE

PR RICHTUNG.AENDERN
 DEF "HILBERT.ANDERS UEBERSETZUNG PRLISTE "HILBERT.ANDERS
ENDE

PR MACHE.WOERTERBUCH :PAARLISTE
 WENN :PAARLISTE = [ ] DANN RUECKKEHR
 SETZE ERSTES ERSTES :PAARLISTE LETZTES ERSTES :PAARLISTE
 MACHE.WOERTERBUCH OHNEERSTES :PAARLISTE
ENDE

PR MACHE.WOERTERBUCH.HILBERT.KURVE
 MACHE.WOERTERBUCH OHNEERSTES PRLISTE "WOERTERBUCH.HILBERT.KURVE
ENDE

PR WOERTERBUCH.HILBERT.KURVE
 RECHTS LINKS
 LINKS RECHTS
ENDE
```

Man braucht zwar etwas Geduld, um das Zeichnen der Hilbertkurve durch diese neue Version zu verfolgen, aber sie soll ja auch nur der Veranschaulichung des Prozesses dienen.

Abschließend wollen wir noch einmal zum Problem der flächenfüllenden Kurve zurückkehren. Um den Grenzprozeß zu veranschaulichen, ist es notwendig, Hilbertkurven mit verschiedener Stufenzahl in dasselbe Quadrat zu zeichnen. Der Parameter :GROESSE muß dabei um so kleiner werden, je größer die Stufenzahl wird. Aber was ist der richtige Wert von :GROESSE ?

Aufgabe 8.8: Es sei A die Seitenlänge des Quadrats, in das die Hilbertkurven gezeichnet werden sollen. Zeigen Sie: Ist die Stufenzahl gleich n, so muß der Parameter :GROESSE beim Aufruf von HILBERT :GROESSE :STUFE :PARITAET den Wert $A / (2^n - 1)$ haben.

Aufgabe 8.9: Schreiben Sie eine "Initialisierungs-Prozedur" AUFRUF.HILBERT :A :STUFE, mit der eine Hilbertkurve der vorgegebenen Stufe genau in ein Quadrat der Seitenlänge :A gezeichnet wird. Schreiben Sie dazu eine Hilfsfunktion ZWEIERPOTENZ :N .

Bei einer etwas höheren Stufenzahl wird der flächenfüllende Charakter der Hilbertkurve schon ziemlich deutlich.

Abbildung 8.13: HILBERT 3 6 1 (Startpunkt (100 / -100))

8.5 Weitere Anwendungen von PRLISTE

Gelegentlich ist es nützlich, beim Lauf einer Prozedur feststellen zu können, ob ein bestimmtes vorgegebenes Wort ein Logo-Grundwort oder ein Prozedurname oder keines von beiden ist. Man bräuchte dazu prädikative Funktionen GRUNDWORT? :WORT beziehungsweise PROZEDURNAME? :WORT, deren Ausgabe, je nach der Eigenschaft von :WORT, entweder WAHR oder FALSCH ist.

Diese prädikativen Funktionen gibt es zwar nicht als Logo-Grundwörter; sie sind aber mit Hilfe der Prozedurliste leicht herzustellen. Die Funktion PRLISTE hat nämlich über die schon besprochene Wirkung hinaus noch die folgenden Eigenschaften:

PRLISTE :X = Prozedurliste von :X, falls :X ein Prozedurname ist;
= :X, falls :X ein Logo-Grundwort ist;
= [ ] sonst.

Die Eingabe :X ist also genau dann ein Logo-Grundwort, wenn sie mit ihrer "Prozedurliste" übereinstimmt.

```
PR GRUNDWORT?  :X
 RUECKGABE  ( :X  =  PRLISTE  :X )
ENDE
```

Einige Beispiele:

GRUNDWORT? "STIFTHOCH
ERGEBNIS: WAHR

GRUNDWORT? "UEBERSETZUNG
ERGEBNIS: FALSCH

GRUNDWORT? "3.14
ERGEBNIS: FALSCH

GRUNDWORT? "+
ERGEBNIS: WAHR

GRUNDWORT? "=
ERGEBNIS: WAHR

GRUNDWORT? "(
ERGEBNIS: WAHR

GRUNDWORT? "%
ERGEBNIS: FALSCH

GRUNDWORT? ERSTES [ERSTES]
ERGEBNIS: WAHR

Wenn man zum Beispiel GRUNDWORT? 4 oder GRUNDWORT? [ERSTES] eingibt, bringt Logo die Fehlermeldung:
PRLISTE MAG 4 NICHT ALS EINGABE beziehungsweise
PRLISTE MAG [ERSTES] NICHT ALS EINGABE .

Aufgabe 8.10: Verbessern Sie die obige Version von GRUNDWORT? so, daß in dem Fall, wo eine Zahl oder eine Liste eingegeben wird, die Fehlermeldung vermieden und FALSCH als Funktionswert ausgegeben wird.

Die Funktion PROZEDURNAME? :X läßt sich mit Hilfe der Prozedurliste folgendermaßen schreiben:

```
PR PROZEDURNAME?  :X
 RUECKGABE ALLE? ( LISTE? PRLISTE  :X ) ( NICHT ( PRLISTE  :X = [ ] ) )
ENDE
```

PROZEDURNAME? "ERSETZE.WORT
ERGEBNIS: WAHR

PROZEDURNAME? "GRUNDWORT?
ERGEBNIS: WAHR

PROZEDURNAME? "PROZEDURNAME?
ERGEBNIS: WAHR

PROZEDURNAME? "WAHR
ERGEBNIS: FALSCH

PROZEDURNAME? "XANTHIPPE
ERGEBNIS: FALSCH

PROZEDURNAME? "*
ERGEBNIS: FALSCH

Die folgende Prozedur fischt alle Prozedurnamen aus der Liste :L heraus:

```
PR PROZEDURNAMEN  :L
 WENN  :L = [ ]   DANN RUECKGABE  :L
 WENN  PROZEDURNAME? ERSTES  :L  DANN  RUECKGABE ______________
 ___  MITERSTEM  (ERSTES  :L )  ( PROZEDURNAMEN  OHNEERSTES  :L)
 RUECKGABE  PROZEDURNAMEN  OHNEERSTES  :L
ENDE
```

Ein Beispiel:

PROZEDURNAMEN [HUGO UEBERSETZUNG XAVER PROZEDURNAME? STIFTAB]
ERGEBNIS: [UEBERSETZUNG PROZEDURNAME?]

Die englischen Logo-Versionen verfügen über eine Funktion .CONTENTS, deren Ausgabe eine Liste mit allen Wörtern ist, die Logo zum Zeitpunkt des Aufrufs kennt. Diese umfaßt alle Grundwörter, alle Variablennamen, alle Namen benutzerdefinierter Prozeduren und sogar alle Tippfehler, solange diese nicht durch eine garbage collection gestrichen wurden. In der deutschen Logo-Version für den Commodore 64 Computer heißt das entsprechende Grundwort ARBEITSLISTE (kurz AL). Allerdings werden in dieser Version sogenannte "truly worthless atoms", wie zum Beispiel Tippfehler, nicht durch garbage collection aus der Arbeitsliste gestrichen.

Aufgabe 8.11: Connie Curzdenker sagt "Ich brauche doch den .CONTENTS-Befehl gar nicht, denn ich kann mir doch mit Hilfe der Befehle ZEIGE TITEL und ZEIGE NAMEN alles zeigen lassen". Setzen Sie sich mit diesem Argument auseinander.

Durch den Aufruf PROZEDURNAMEN ARBEITSLISTE (englisch: PROCEDURE-NAMES .CONTENTS) können wir uns also eine vollständige Liste aller zum Zeitpunkt des Aufrufs bekannten Prozedurnamen verschaffen. (Bei Verwendung einer englischsprachigen Logo-Version ist natürlich vorher die Funktion PROZEDURNAMEN ins Englische zu übersetzen).

Eine weitere nützliche Anwendung von PRLISTE ist die Möglichkeit, festzustellen, ob eine bestimmte benötigte Prozedur überhaupt vorhanden ist und sie gegebenenfalls von einer Diskette nachzuladen. Dies erweist sich besonders in den Fällen als nützlich, wenn man eine umfangreiche Prozedur nur selten benutzt. Man kann sie dann vor der Ausführung nachladen und nach der Ausführung wieder vergessen, um dadurch Speicherplatz zu gewinnen.

Ich habe diese Methode zum Beispiel im folgenden Zusammenhang praktiziert: Beim Zeichnen von Funktionsschaubildern und Histogrammen benötigt man natürlich eine Prozedur ACHSEN, welche die Koordinatenachsen an der richtigen Stelle auf den Bildschirm zeichnet. Manchmal braucht man die Achsen skaliert, manchmal unskaliert. Je nach Bedarf sollte ACHSEN auf eine Hilfsprozedur SKALIERUNG zugreifen können.

Die Skalierung von Koordinatenachsen ist zwar kein Hexenwerk, wenn man es aber richtig machen will, und dazu gehört zum Beispiel auch die Festlegung optimaler Einheiten (natürlich in geeigneten Vielfachen von Zehnerpotenzen), so wird daraus leicht eine sehr umfangreiche Prozedur, die man für jedes Schaubild höchstens einmal, manchmal aber auch gar nicht, benutzt. Es bietet sich also an, mit der Prozedur SKALIERUNG folgendermaßen zu verfahren: Wir speichern diese

Prozedur (bzw. die entsprechende Prozedurgruppe) zunächst unter dem Namen SKALIERUNGSDATEI auf einer Diskette ab. In der Prozedur ACHSEN fragen wir im Falle des Skalierungswunsches sicherheitshalber vorher ab, ob die Prozedur SKALIERUNG überhaupt geladen ist, bevor wir sie auszuführen versuchen. In der folgenden Prozedurskizze ist SKALIERUNG? ein Parameter, mit dem man festlegen kann, ob die Achsen überhaupt skaliert werden sollen oder nicht. (Bei :SKALIERUNG = WAHR wird skaliert, sonst nicht).

```
PR ACHSEN  Parameter von Achsen, darunter  :SKALIERUNG?
  ...
  Befehle zum Zeichnen der Achsen
  ...
  WENN  :SKALIERUNG? ____________________________________________
  ____    DANN  WENN   PRLISTE "SKALIERUNG  =  [ ] _______________
  __________        DANN  LADE  "SKALIERUNGSDATEI ________________
  ____      SKALIERUNG  Parameter von Skalierung ________________
  ____      VERGISS SKALIERUNG
ENDE
```

Aufgabe 8.12: (Ein Graphik-Projekt)
Ergänzen Sie Ihre Prozeduren zum Zeichnen von Funktionsschaubildern und Histogrammen in der oben skizzierten Form durch die Hilfsprozeduren ACHSEN und SKALIERUNG. Vervollständigen Sie dazu diese Prozeduren.

Kapitel 9: Einige nützliche Dienstprogramme

In diesem Kapitel sind einige Hilfsprogramme zusammengestellt, die man gelegentlich (manchmal dringend) benötigt. Die Prozeduren sind sparsamer kommentiert als diejenigen in den vorangegangenen Kapiteln. Denn zum Erwerb einer Grundkompetenz im Programmieren gehört auch die Fähigkeit, vorliegende Programme analysieren, verstehen und seinen eigenen Bedürfnissen entsprechend verändern zu können. Und diese Fähigkeit läßt sich an den folgenden Dienstprogrammen gut trainieren.

Der Quicksort ist ein weiteres, sehr lehrreiches Beispiel zum Themenbereich *rekursive Listenverarbeitung*. Die Pretty-Print-Umgebung dient nochmals der Illustration des mächtigen Prozedurlisten-Konzepts. Besonders in den Prozeduren der Drucke-Stellengerecht-Umgebung und der Pretty-Print-Umgebung steckt sehr viel mehr "Ausgabekosmetik" als in den bisherigen Prozeduren. Dennoch sind auch die vorliegenden Prozeduren gerade im Kosmetik-Bereich noch erheblich verbesserungsfähig; die Ausschmückung sei dem Leser zur Übung überlassen.

Im Gegensatz zur bisher geübten Praxis werden die Logo-Grundwörter im folgenden meist in ihrer Kurzform verwendet.

9.1 Quicksort

Häufig muß man willkürlich vorliegende Daten in eine sortierte Form bringen. Das **Sortieren** ist ein wichtiges Teilgebiet der Informatik. Nur in wenigen Spezialbereichen gibt es derart viele vergleichsweise gut untersuchte Algorithmen wie für das Sortieren. Der bekannte Informatiker Donald Knuth hat dem Thema "Sorting and Searching" in seinem monumentalen Werk "The Art of Computer Programming" einen 723-seitigen Teilband gewidmet.

Ein insgesamt sehr schneller Sortieralgorithmus ist, wie schon der Name sagt, der sogenannte Quicksort. Die mathematische Diskussion der Laufzeiteffizienz des Quicksort würde den Rahmen dieses Buches überschreiten. Der interessierte Leser sei auf das obengenannte Buch von D.Knuth oder zum Beispiel auf die Bücher "Fundamentals of Data Structures" (E.Horowitz / S.Sahni) und "Algorithmen und Datenstrukturen" (N.Wirth) verwiesen; siehe Literaturverzeichnis.

Die Funktion QUICKSORT nimmt als aktuellen Eingabeparameter eine Liste auf. Sie gibt die Liste in (alphabetisch) sortierter Form als Funktionswert zurück.

```
QUICKSORT   [STUTTGART FREIBURG HABMURG MUENCHEN BERLIN ___
            FRANKFURT ]
_________
ERGEBNIS:   [BERLIN FRANKFURT FREIBURG HAMBURG MUENCHEN ___
            STUTTGART ]
_________
```

QUICKSORT arbeitet nach der folgenden Grundidee: Aus der gegebenen Liste wird (willkürlich) ein *Trennelement* :T herausgegriffen. In unserem Falle ist es einfach das erste Element der Liste :L. Die zu sortierende Liste wird durch dieses Trennelement in drei Teillisten zerlegt: UNTERHALB :T :L enthält (in unsortierter Form) alle Elemente von :L, die dem Sortierbegriff nach kleiner als das Trennelement sind, GLEICH :T :L enthält alle Elemente von :L, die gleich dem Trennelement sind und

OBERHALB :T :L enthält die Elemente, welche größer als das Trennelement sind. Auf UNTERHALB :T :L und OBERHALB :T :L wird nun wiederum QUICKSORT angewandt. Schließlich werden die drei sortierten Teillisten

QUICKSORT UNTERHALB :T :L
GLEICH :T :L und
QUICKSORT OBERHALB :T :L

durch die Logo-Grundfunktion SATZ zu einer sortierten Gesamtliste zusammengefügt.

Hier die Prozeduren:

```
PR QUICKSORT :L
 WENN :L = [ ] RG :L
 RG ( SATZ ( QUICKSORT UNTERHALB ER :L :L ) ___________
 ___ ( GLEICH ER :L :L ) ( QUICKSORT OBERHALB ER :L :L ) )
ENDE

PR UNTERHALB :T :L
 WENN :L = [ ] RG :L
 WENN ( KLEINER? ER :L :T ) ( RG ME ER :L UNTERHALB :T OE :L )
 RG UNTERHALB :T OE :L
ENDE

PR GLEICH :T :L
 WENN :L = [ ] RG :L
 WENN ( :T = ER :L ) ( RG ME :T GLEICH :T OE :L )
 RG GLEICH :T OE :L
ENDE

PR OBERHALB :T :L
 WENN :L = [ ] RG :L
 WENN ( GROESSER? ER :L :T ) ( RG ME ER :L OBERHALB :T OE :L )
 RG OBERHALB :T OE :L
ENDE
```

Die folgenden prädikativen Funktionen KLEINER? und GROESSER? vergleichen die eingegebenen Wörter *lexikographisch* anhand der ASCII-Werte ihrer "Buchstaben". (Auch Sonderzeichen und Ziffern werden als Buchstaben angesehen). Falls von vorn herein Zahlen vorliegen, werden sie im arithmetischen Sinne verglichen. Wenn andere Sortierkriterien erwünscht sind, sind in der Quicksort-Umgebung nur diese beiden Funktionen entsprechend abzuändern.

```
PR KLEINER? :X :Y
 WENN ( ALLE? ZAHL? :X ZAHL? :Y ) RG :X < :Y
 WENN :X = :Y RG "FALSCH
 WENN LEER? :X RG "WAHR
 WENN LEER? :Y RG "FALSCH
 WENN ASCII ER :X < ASCII ER :Y RG "WAHR
 WENN ASCII ER :X > ASCII ER :Y RG "FALSCH
 RG KLEINER? OE :X OE :Y
ENDE
```

```
PR GROESSER?  :X  :Y
 WENN  :X  =  :Y   RG  "FALSCH
 WENN  KLEINER?  :X  :Y   RG  "FALSCH
 RG  "WAHR
ENDE
```

Das Prüfwort LEER? ist in Kapitel 4, Abschnitt 4.7 beschrieben.

Alle Funktionen dieser Quicksort-Umgebung können autonom aufgerufen und getestet werden.

9.2 Engels Universalalgorithmus für den Logarithmus und die Arkusfunktionen

Arthur Engel ist es gelungen, eine ganze Reihe vornehmlich trigonometrischer Funktionen mit Hilfe des folgenden verblüffend einfachen Algorithmus zu approximieren.

```
PR UNIVERSAL.ALGORITHMUS  :A  :B  :C
 SOLANGE [ ABS ( :A - :B )  >  0.005 ]
         [ SETZE  "A  ( :A + :B ) / 2   SETZE  "B  QW ( :A * :B ) ]
 RG  ( 3 * :C ) / ( 2 * :B + :A )
ENDE
```

Allein durch geeignete Interpretation der Eingabeparameter lassen sich zum Beispiel die folgenden Funktionen annähern. (Damit der Kontrollfluß nicht gestört wird, ist in den nachfolgenden Funktionen auch die Fehlermeldung bei unzulässigem Argument als Funktionswertrückgabe realisiert; man vergleiche Kapitel 5, Abschnitt 5.2: lineare und quadratische Gleichungen). Die Hilfsprozeduren und -funktionen ABS, EXP, PI und SOLANGE sind (in dieser Reihenfolge) in den Abschnitten 5.2, 6.3, 6.6.2 und 7.1 zu finden.

```
PR LN  :X
 WENN  NICHT :X > 0  RG  "UNDEFINIERT
 RG  UNIVERSAL.ALGORITHMUS  ( 1 + :X ) / 2   ( QW :X )   ( :X - 1 )
ENDE

PR ARCSIN  :X
 WENN  NICHT  ( ABS :X ) <  1  RG  "UNDEFINIERT
 RG  UNIVERSAL.ALGORITHMUS  ( QW ( 1 - :X * :X ) )   1   :X
ENDE

PR ARCCOS  :X
 WENN  NICHT ( ABS :X ) < 1  RG  "UNDEFINIERT
 RG  UNIVERSAL.ALGORITHMUS  :X   1   QW ( 1 - :X * :X )
ENDE

PR ARC.TAN  :X
 RG  UNIVERSAL.ALGORITHMUS  1  ( QW ( 1 + :X * :X ) )  :X
ENDE
```

```
PR ARCCOT  :X
 RG  (PI / 2 ) -  ARC.TAN  :X
ENDE
```

Beim Arkustangens wurde ein Punkt eingefügt, weil es diese Funktion (ohne Punkt) auch als Logo-Grundfunktion gibt.

Beispiele:

LN 2
ERGEBNIS: 0.693147

ARCSIN 0.3
ERGEBNIS: 0.304693

ARCCOS (-0.6)
ERGEBNIS: 2.21429

ARC.TAN 5
ERGEBNIS: 1.3734

ARCCOT 2
ERGEBNIS: 0.46365

Die *Logarithmusfunktion* ist die Umkehrfunktion der *Exponentialfunktion*; die *Arkusfunktionen* sind die Umkehrungen der entsprechenden trigonometrischen Funktionen. Ist allgemein gesprochen g die **Umkehrfunktion** der Funktion f, so gilt für alle x aus dem jeweils zulässigen Definitionsbereich (in mathematischer Schreibweise):

$$g(f(x)) = x \quad \text{und}$$
$$f(g(x)) = x$$

Diese Eigenschaft können wir für einen eleganten Genauigkeitstest ausnutzen:

EXP LN 3
ERGEBNIS: 3.0

LN EXP (-5)
ERGEBNIS: -4.99998

Um einen entsprechenden Test mit den Arkusfunktionen durchführen zu können, müssen wir uns zunächst noch trigonometrische Funktionen definieren, bei denen der Winkel im *Bogenmaß* und nicht, wie bei den Logo-Grundfunktionen, im *Gradmaß* einzugeben ist. Die Umwandlung vomGradmaß ins Bogenmaß und umgekehrt beruht auf der Verhältnisgleichung

$$\frac{\text{Winkel im Gradmaß}}{\text{360 Grad}} = \frac{\text{Winkel im Bogenmaß (am Einheitskreis)}}{2 * \text{PI}}$$



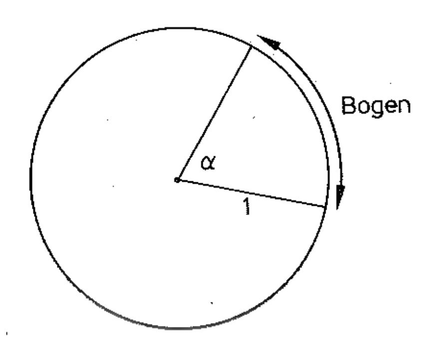


Abbildung 9.1: Umwandlung vom Gradmaß ins Bogenmaß

Die folgenden beiden Funktionen übernehmen die Umwandlung vom Gradmaß ins Bogenmaß und umgekehrt; man vergleiche mit Kapitel 2 (Abschnitt 2.4.4.3).

```
PR BOGENMASS  :W
 RG  :W * PI / 180
ENDE

PR GRADMASS  :B
 RG  :B * 180 / PI
ENDE
```

Mit Hilfe dieser Funktionen und den in Logo eingebauten Grundfunktionen SIN und COS können wir nun leicht die trigonometrischen Funktionen für das Bogenmaß definieren. (Die kurzen Namen gelten jeweils für das Gradmaß, die langen für das Bogenmaß).

```
PR SINUS  :X
 RG  SIN GRADMASS :X
ENDE

PR COSINUS  :X
 RG  COS  GRADMASS  :X
ENDE

PR TAN  :W
 WENN  ( COS :W )  = 0   RG  "UNDEFINIERT
 RG  ( SIN :W ) / ( COS :W )
ENDE

PR TANGENS  :X
 RG TAN GRADMASS :X
ENDE

PR COT  :W
 WENN  ( SIN :W )  = 0   RG  "UNDEFINIERT
 RG  ( COS :W ) / ( SIN :W )
ENDE
```

```
PR COTANGENS  :X
 RG  1 / TANGENS :X
ENDE
```

Hier einige Proben:

ARCSIN SINUS 0.5
ERGEBNIS: 0.499981

SINUS ARCSIN 0.8
ERGEBNIS: 0.799985

ARCCOS COSINUS 1.5
ERGEBNIS: 1.5

COSINUS ARCCOS (-0.3)
ERGEBNIS: -0.299986

ARC.TAN TANGENS 0.75
ERGEBNIS: 0.75 (*)

TANGENS ARC.TAN 3
ERGEBNIS: 3.0 (* *)

Die eingebaute Logo-Grundfunktion ARCTAN erwartet zwei Eingabewerte x und y und gibt den Arkustangens von x/y als Winkel im Gradmaß zwischen 0 Grad und 360 Grad zurück. Nach entsprechender Anpassung auf das Bogenmaß erhalten wir mit dieser Funktion folgende Werte:

BOGENMASS ARCTAN TANGENS 0.75 1
ERGEBNIS: 0.749991 (* * *)

TANGENS BOGENMASS ARCTAN 3 1
ERGEBNIS: 2.99944 (* * * *)

Für die Analyse des Engelschen Universalalgorithmus sei auf das Buch "Elementarmathematik vom algorithmischen Standpunkt" von A.Engel, besonders Abschnitt 3.5, verwiesen. Der Universalalgorithmus ist zwar sehr effizient, aber wegen der allgemein sehr geringen Genauigkeit der Gleitkommaarithmetik von 8-Bit-Computern sind präzise Aussagen über die Zuverlässigkeit der durch obige Funktionen ermittelten Werte kaum möglich. Die mit dem Universalalgorithmus ermittelten Funktionswerte scheinen insgesamt genauer zu sein als die mit Hilfe der eingebauten Funktionen ermittelten Werte (man vergleiche (*) mit (* * *) und (* *) mit (* * * *)). Allerdings sind die eingebauten Funktionen schneller, da sie auf der Maschinenebene implementiert sind.

Der Universalalgorithmus kann auch zur Approximation der "Areafunktionen" ARSINH, ARCOSH, ARTANH und ARCOTH herangezogen werden (siehe Engel, Seite 73 und 76). Ein informeller Test hat jedoch relativ schlechte Werte ergeben (was nicht am Algorithmus sondern an der Beschränktheit der Zahldarstellung in den meisten 8-Bit-Computern liegt).Deshalb wurde hier auf die Behandlung dieser Funktionen verzichtet.

Aufgabe 9.1: Ändern Sie den Engelschen Universalalgorithmus so ab, daß die in der obigen Version fest verankerte Toleranzschwelle (0.005) variabel eingegeben werden kann und untersuchen Sie, bei Bedarf auch für die Areafunktionen, Laufzeit und Genauigkeit bei unterschiedlichen Toleranzwerten.

9.3 Stellengerechtes Drucken

Die folgenden Prozeduren dienen insgesamt dazu, die vier Hauptprozeduren

- DRRB für drucke rechtsbündig,
- DZRB für drucke rechtsbündig mit nachfolgendem carriage return,
- DRSG für drucke stellengerecht und
- DZSG für drucke stellengerecht mit nachfolgendem carriage return

zu unterstützen. Die Prozeduren DRRB und DZRB haben zwei Parameter, das zu druckende Wort und eine ganze Zahl, welche die zur Verfügung stehende Gesamtlänge des Druckfeldes angibt. Als "Worte" sind natürlich insbesondere auch Zahlen zugelassen. DRSG und DZSG haben jeweils drei Parameter: eine zu druckende Zahl, die Gesamtlänge des Druckfeldes und die Anzahl der Nachkommastellen.

Die vier obengenannten Prozeduren stimmen inhaltlich also weitgehend mit den write- bzw. writeln-Befehlen von Pascal überein.

Das auszudruckende Objekt wird mit Hilfe der Funktion STELLENMASKE zunächst vollständig in formatierter Form als Wort aufbereitet. Es wurde bewußt darauf verzichtet, die Befehle zur Cursor-Positionierung zu verwenden. Denn diese Methode wäre stark vom benutzten Computer, ja sogar von der benutzten Logo-Version abhängig und somit weit weniger zur Übertragung auf andere Geräte oder Logo-Versionen geeignet gewesen. Außerdem würde sie sich nur zum Ausdruck auf dem Bildschirm eignen. Mit der im folgenden praktizierten Methode kann man aber auch formatierte Ausdrucke auf Papier oder auf Diskette produzieren.

Hier zunächst die Hauptprozeduren:

```
PR DRRB  :WORT  :GESAMTLAENGE
 ;  DRUCKE RECHTSBUENDIG
 PRUEFE  :GESAMTLAENGE  >  ( LAENGE :WORT )
 WENNFALSCH  DR  :WORT  RUECKKEHR
 DR  LEERWORT  ( :GESAMTLAENGE - LAENGE :WORT )  DR  :WORT
ENDE
```

(Die Funktion LEERWORT :N wird weiter unten erklärt. Sie ist eine leichte Verallgemeinerung der entsprechenden konstanten Funktion aus Kapitel 4, Abschnitt 4.2.4).

```
PR DZRB  :WORT  :GESAMTLAENGE
 ;  DRUCKEZEILE RECHTSBUENDIG
 DRRB  :WORT  :GESAMTLAENGE
 DR  ZEICHEN 13    ; CARRIAGE RETURN ZEICHEN
ENDE
```

```
PR DRSG  :X  :GESAMTLAENGE  :NACHKOMMASTELLEN
 ; DRUCKE STELLENGERECHT
 DRRB ( STELLENMASKE  :X  :NACHKOMMASTELLEN )  :GESAMTLAENGE
ENDE

PR DZSG  :X  :GESAMTLAENGE  :NACHKOMMASTELLEN
 ; DRUCKEZEILE STELLENGERECHT
 DRSG  :X  :GESAMTLAENGE  :NACHKOMMASTELLEN
 DR  ZEICHEN 13    ; CARRIAGE RETURN ZEICHEN
ENDE
```

Wenn das auszudruckende Objekt länger ist als die angegebene Gesamtlänge, so wird es einfach mit dem normalen Logo-Druckbefehl gedruckt. Das eigentliche "Arbeitspferd" der Drucke-Stellengerecht-Umgebung ist die Funktion STELLENMASKE. Sie dient der Aufbereitung der auszudruckenden Zahl. Es ist nicht die schlechteste Methode, im interaktiven Betrieb etwas mit dieser Funktion zu experimentieren, um ihre Wirkungsweise besser verstehen zu lernen.

```
PR STELLENMASKE  :X  :N
 LOKAL  "MASKE
 SETZE  "MASKE  RUNDE ( :X * POT 10 :N )
 PRUEFE  LAENGE  :MASKE  <  :N
 WW  SETZE  "MASKE  ( WORT  ",  MIT.FUEHRENDEN.NULLEN :MASKE :N )
 WF  SETZE  "MASKE  ( WORT  ANFANGSSTUECK  :MASKE ____________________
 ________  ( ( LAENGE :MASKE ) - :N )  ",  ENDSTUECK  :MASKE  :N )
 WENN  ER  :MASKE  =  ",  SETZE  "MASKE  (  WORT  "0  :MASKE )
 WENN  LZ  :MASKE  =  ",  SETZE  "MASKE  OL  :MASKE
 RG  :MASKE
ENDE
```

Die Funktion POT :X :N liefert die :N-te Potenz von :X.

```
PR POT  :X  :N
 WENN  :N  =  0  RG  1
 RG  :X  *  POT  :X  ( :N - 1 )
ENDE
```

Hier die restlichen Prozeduren:

```
PR LAENGE  :X
 WENN  :X  =  "  RG  0
 RG  1 + LAENGE  OE  :X
ENDE

PR MIT.FUEHRENDEN.NULLEN  :X  :GESAMTLAENGE
 WENN  NICHT  ( :GESAMTLAENGE  >  LAENGE  :X ) RG  :X
 RG  MIT.FUEHRENDEN.NULLEN  ( WORT  "0  :X )  :GESAMTLAENGE
ENDE

PR ANFANGSSTUECK  :X  :N
 WENN  :X  =  "  RG  :X
 WENN  :N  =  0  RG  "
 RG  ( WORT  ( ER :X )  ANFANGSSTUECK  OE  :X  :N - 1 )
ENDE
```

```
PR ENDSTUECK  :X  :N
 WENN  :X  =  "   RG  :X
 WENN  :N  =  0   RG  "
 RG  ( WORT  ( ENDSTUECK  OL  :X  :N - 1 )  LZ  :X )
ENDE

PR LEERWORT  :N
 WENN  :N  =  0   RG  "
 RG  ( WORT  ZEICHEN 32  LEERWORT  :N - 1 )
ENDE
```

Die nachfolgende Prozedur DEMO sollte zu dem unten wiedergegebenen Ausdruck führen.

```
PR DEMO
 DZ  [ STELLENGERECHTER AUSDRUCK ]
 DZ  "BEISPIELE:
 DZ  "
 DRRB  "DEMO  10  DRRB  "ANFANG  15
 DZ  "
 DZSG  ( 123 * SIN 987 )  15  5
 DZSG  ( SIN 0.25  )  20  10
 DZSG  1/16  18  8
 DZSG  1/80  18  8
 DZSG  1/125  18  8
 DZSG  1234  *  2345  12  2
 DZSG  987  /  ( - 2 )  14  4
 DZSG  10000 / 111  13  3
 DZSG  7 / 2   9   0
 DRSG  ( 1.25 * 1.25 )  15  5
 DRRB  "DEMO  8
 DRRB  "ENDE  6
ENDE
```

```
STELLENGERECHTER AUSDRUCK
BEISPIELE
          DEMO           ANFANG
        -122,83142
            0,0043620352
            0,06250000
            0,01250000
            0,00800000
   2893730,00
        -493,5000
           90,090
            4
            1,56250     DEMO  ENDE
```

Aufgabe 9.2: In den Prozeduren der Drucke-Stellengerecht-Umgebung wird das deutsche Dezimalkomma an Stelle des englischen Dezimalpunktes benutzt. Erweitern Sie die Prozeduren so, daß auf Wunsch des Benutzers entweder das Dezimalkomma oder der Dezimalpunkt verwendet wird.

Aufgabe 9.3: Erweitern Sie die obigen Prozeduren so, daß Geldbeträge auf Wunsch mit Scheckschutz-Zeichen ausgedruckt werden können.

9.4 Pretty Printing

Mit den überlaufenden Zeilen sehen die im Bildschirm-Editor stehenden Logo-Prozeduren oft nicht besonders übersichtlich aus; dies besonders dann, wenn auf dem Bildschirm nur 40 Zeichen pro Zeile dargestellt werden.

Deshalb wurde in diesem Buch eine übersichtlichere Form der Darstellung gewählt. Mit Hilfe des Unterstreichungszeichens wurden die Prozeduren in optisch besser strukturierter Form wiedergegeben. Man bezeichnet diese Darstellungsform gelegentlich auch als pretty printing.

Das pretty printing wurde zunächst von Hand durchgeführt. Da das Verfahren nach ziemlich klaren Regeln abläuft (siehe auch Bemerkung 1. in Abschnitt 2.3) liegt es nahe, diesen Prozeß zu automatisieren. Das Ergebnis dieser Überlegungen ist die folgende Pretty-Print-Umgebung.

```
PR PRETTY.PRINT.PROZEDUR  :PN  :LR  :RR
 ; PN  =  PROZEDURNAME
 ; LR  =  LINKER RAND
 ; RR  =  RECHTER RAND
 PRETTY.PRINT.LISTE  ( SATZ  "PR  :PN  ER  PRLISTE  :PN )  :LR  :RR  1
 PRETTY.PRINT.LISTE  OE  PRLISTE  :PN  ( :LR + 1 )  :RR  0
 PRETTY.PRINT.LISTE  ( LISTE  "ENDE )  :LR  :RR  1
 CARRIAGE.RETURN
ENDE

PR PRETTY.PRINT.LISTE  :L  :LR  :RR  :STUFE
 LOKAL  "ZREST          ; ZEILENREST
 SETZE  "ZREST  :RR - :LR
 PRETTY.PRINT  :L  :STUFE
ENDE

PR PRETTY.PRINT  :L  :STUFE
 WENN  :STUFE  >  0   NEUE.ZEILE
 WENN  :STUFE  >  1   WORT.DRUCKEN  "'['  "FALSCH
 PRETTY.PRINT.1  :L  :STUFE
 WENN  :STUFE  >  1   WORT.DRUCKEN  "']'  "WAHR
ENDE

PR  PRETTY.PRINT.1  :L  :STUFE
 WENN  :L  =  [ ]   RK
 PRUEFE  LISTE?  ERSTES  :L
 WW  PRETTY.PRINT  ERSTES  :L  :STUFE + 1
 WF  WORT.DRUCKEN  ERSTES  :L  ( ( ER  :L )  =  ( LZ  :L ) )
 PRETTY.PRINT.1  OE  :L  :STUFE
ENDE
```

In den Prozeduren PRETTY.PRINT und PRETTY.PRINT.1 tritt die Rekursion in ver-

schränkter Form auf: PRETTY.PRINT ruft PRETTY.PRINT.1 auf und diese Prozedur greift ihrerseits wieder auf PRETTY.PRINT zurück, u.s.w. .

```
PR WORT.DRUCKEN  :X  :LZ?
 ; LZ?  =  IST  :X  LETZTES ELEMENT DER LISTE?
 PRUEFE  ( ALLE?  ( :STUFE = 1 )  :LZ? )
 WW  DANN WENN ( LAENGE :X )  >  :ZREST  DANN  ZEILEN.UEBERLAUF
 WF  DANN WENN ( LAENGE :X )  >  ( :ZREST - EINRUECKUNG - 1 ) ________
 _____    DANN ZEILEN.UEBERLAUF
 DR  :X
 DR  LEERZEICHEN
 SETZE  "ZREST  :ZREST - ( LAENGE :X ) - 1
ENDE

PR NEUE.ZEILE
 WENN ( ALLE? ( :STUFE > 1 )  ( :ZREST < :RR - :LR ) ) ________
 ___  DANN  FUELLDRUCK  :ZREST  UNTERSTREICHUNG
 CARRIAGE.RETURN
 FUELLDRUCK  :LR  LEERZEICHEN
 SETZE  "ZREST  :RR - :LR
 WENN  :STUFE > 0  DANN ________________________________
 ___ FUELLDRUCK (( :STUFE - 1 ) * EINRUECKUNG) UNTERSTREICHUNG ______
 ___ SETZE  "ZREST  ( :ZREST - ( :STUFE - 1 ) * EINRUECKUNG )
ENDE

PR ZEILEN.UEBERLAUF
 FUELLDRUCK  :ZREST  UNTERSTREICHUNG
 CARRIAGE.RETURN
 FUELLDRUCK  :LR  LEERZEICHEN
 FUELLDRUCK  :STUFE * EINRUECKUNG  UNTERSTREICHUNG
 SETZE  "ZREST  ( :RR - :LR - :STUFE * EINRUECKUNG )
ENDE

PR FUELLDRUCK  :N  :Z
 WH  :N  [ DR  :Z ]
ENDE

PR EINRUECKUNG
 RG  2
ENDE

PR CARRIAGE.RETURN
 DR  ZEICHEN  13
ENDE

PR LEERZEICHEN
 RG  ZEICHEN  32
ENDE

PR UNTERSTREICHUNG
 RG  ZEICHEN  95
ENDE
```

Die Hilfsprozeduren LAENGE, LEER? und SOLANGE wurden bereits früher behandelt.

Hier zur Demonstration zwei Tilgungsversionen: zunächst im 40-Zeichen-pro-Zeile Editor Format und anschließend im "pretty print look". Die Sternchen sollen den Bildschirmrand andeuten.

1. Beispiel:

```
******************************************
*PR TILGUNG.LOGO.LIKE :DO :P :A          *
* TILGUNG.LOGO.LIKE.1 :DO :P :A 0        *
*ENDE                                    *
*                                        *
*PR TILGUNG.LOGO.LIKE.1 :D :P :A :K      *
* DR :K                                  *
* WH 5 [DR LEERZEICHEN]                  *
* DZ :D                                  *
* WENN :D > 0 DANN TILGUNG.LOGO.LIKE.1 (!*
* :D * ( 1 + :P / 100 ) - :A ) :P :A :K !*
*+ 1                                     *
*ENDE                                    *
*                                        *
******************************************
 1234567890123456789012345678901234567890
          1         2         3         4
```

2. Bespiel:

```
******************************************
*PR TILGUNG.PASCAL.LIKE                  *
* DRUCKE "ANFANGSDARLEHEN:               *
* SETZE "DO ERSTES EINGABE               *
* DRUCKE "ZINSSATZ:                      *
* SETZE "P ERSTES EINGABE                *
* DRUCKE "ANNUITAET:                     *
* SETZE "A ERSTES EINGABE                *
* SETZE "K 0                             *
* SETZE "D :DO                           *
* SOLANGE [:D>0] [SETZE "K :K + 1 SETZE !*
*"D ( 1 + :P / 100 ) * :D - :A DR :K WH !*
*5 [DR LEERZEICHEN] DZ :D]               *
*ENDE                                    *
*                                        *
******************************************
 1234567890123456789012345678901234567890
          1         2         3         4
```

Man beachte insbesondere die Ausrufezeichen in Spalte 40 bei Zeilenüberlauf.

Die Pretty-Print-Darstellung der ersten Tilgungsversion wird nun durch die folgenden Aufrufe erreicht:

PRETTY.PRINT.PROZEDUR "TILGUNG.LOGO.LIKE 2 38
PRETTY.PRINT.PROZEDUR "TILGUNG.LOGO.LIKE.1 2 38

Das Ergebnis dieser Aufrufe sieht so aus:

```
*****************************************
*  PR TILGUNG.LOGO.LIKE :DO :P :A        *
*   TILGUNG.LOGO.LIKE.1 :DO :P :A  0     *
*  ENDE                                  *
*                                        *
*  PR TILGUNG.LOGO.LIKE.1 :D :P :A :K    *
*   DR :K                                *
*   WH 5 ______________________________  *
*   __[ DR LEERZEICHEN ]                 *
*   DZ :D                                *
*   WENN :D > 0 DANN _________________   *
*   __TILGUNG.LOGO.LIKE.1 ( :D * ( 1 __  *
*   __+ :P / 100 ) - :A ) :P :A :K + 1   *
*  ENDE                                  *
*                                        *
*****************************************
 1234567890123456789012345678901234567890
          1         2         3         4
```

Abschließend noch die zweite Tilgungsversion im pretty-print-look:
PRETTY.PRINT.PROZEDUR "TILGUNG.PASCAL.LIKE 2 38

```
*****************************************
*  PR TILGUNG.PASCAL.LIKE                *
*   DRUCKE "ANFANGSDARLEHEN:             *
*   SETZE "DO ERSTES EINGABE             *
*   DRUCKE "ZINSSATZ:                    *
*   SETZE "P ERSTES EINGABE              *
*   DRUCKE "ANNUITAET:                   *
*   SETZE "A ERSTES EINGABE              *
*   SETZE "K 0                           *
*   SETZE "D :DO                         *
*   SOLANGE ___________________________  *
*   __[ :D>0 ] ________________________  *
*   __[ SETZE "K :K + 1 SETZE "D ( 1 __  *
*   ____+ :P / 100 ) * :D - :A DR :K __  *
*   ____WH 5 __________________________  *
*   ____[ DR LEERZEICHEN ] DZ :D ]       *
*  ENDE                                  *
*                                        *
*****************************************
 1234567890123456789012345678901234567890
          1         2         3         4
```

Zur Demonstration wurde in den obigen Beispielen eine relativ kleine Zeilenbreite verwendet. Der Ausdruck wird optisch ansprechender, wenn man beim Ausdruck auf einem Drucker mit einer größeren Zeilenbreite arbeitet. (Das Kommando zur Öffnung des Druckerkanals lautet im MIT Logo für den Apple II Computer AUSGANG 1 beziehungsweise AUSGANG :SLOT, wo :SLOT die Nummer des benutzten *Interface-Steckplatzes* ist).

Leerzeichen, die in Zeichenketten (Wörter) eingebettet sind, vertragen sich nicht sehr gut mit der Syntax von Lisp bzw. Logo. In den obigen Pretty-Print-Prozeduren blieben Wörter mit eingebetteten Leerzeichen zunächst unberücksichtigt. Erweitern Sie die obigen Prozeduren bei Bedarf so, daß auch Wörter, die Leerzeichen enthalten, verarbeitet werden können.

Auch sonst mag Ihnen vielleicht dies oder jenes an der Form des Ausdruckes noch nicht gefallen. Das Schöne an der "offenen" Programmierumgebung von Lisp/Logo ist, daß sich solche Dinge sehr gut nach den Wünschen des Benutzers umgestalten lassen. Wenn Ihnen also das pretty printing in der vorliegenden Form noch nicht zusagt, dann modifizieren Sie es, bis es Ihren Vorstellungen genau entspricht.

Das in diesem Buch praktizierte pretty printing ist allerdings meist etwas freier durchgeführt worden als dasjenige, welches mit Hilfe von PRETTY.PRINT.PROZEDUR zustande kommt.

Alphabetische Liste der Logo-Grundwörter

Im folgenden sind die englischen und deutschen Logo-Grundwörter in alphabetischer Reihenfolge aufgeführt. In der linken Spalte steht jeweils das englische Grundwort für den Apple II Computer in der Terrapin-Syntax oder für den Commodore 64 Computer. In der rechten Spalte steht das entsprechende deutsche Grundwort in der IWT-Syntax. Es gibt eine eigenständige IWT-Version für den Apple II Computer und einen IWT-Sprachzusatz zum Commodore 64 Logo, mit dem die deutsche Syntax implementiert werden kann. Bei den meisten Grundwörtern gibt es keinen Unterschied zwischen dem Apple und dem Commodore Logo. (Die Logo-Versionen weisen einen erheblich höheren Grad an syntaktischer Verträglichkeit auf als die BASIC-Versionen dieser Computer). Falls es ein bestimmtes Grundwort nur in der Apple- oder nur in der Commodore-Version gibt, ist dies in der rechten Spalte angezeigt. Das Commodore Logo kam später heraus als das Apple Logo und enthält deshalb mehr Grundwörter. Sie betreffen besonders den Bereich der Graphik einschließlich der "sprites". Inzwischen gibt es unter der Versionsnummer 2.0 auch eine Weiterentwicklung des englischen Logo von Terrapin, welches die folgenden Grundwörter enthält: COUNT, EMPTY?, ITEM, LOCAL, MEMBER?, SETDISK. Leider gibt es zur Zeit noch keine entsprechende deutsche Version. Erweiterte Logo-Versionen für den Apple II e (128 K) gibt es von LCSI und im Laufe des Jahres 1985 auch von Terrapin.

Neben dem vollen Namen steht jeweils die Abkürzung, falls es eine gibt.

.ASPECT	.SKALA	
.BPT	.BPT	(Apple II)
.CALL	.RUFE	
.CONTENTS	ARBEITSLISTE AL	(Commodore 64)
.CTYO	CTYO	(Commodore 64)
.DEPOSIT	.LEGE	
.EXAMINE	.HOLE	
.GCOLL	.GCOLL	
.NODES	.KNOTEN	
.OPTIONS	OPTION	(Commodore 64)
.SPRINT		(Commodore 64)
ALL	ALLES	
ALLOF	ALLE?	
ANYOF	EINES?	
ASCII	ASCII (gel.: ASC)	(Apple II)
	ASCII	(Commodore 64)
ATAN	ARCTAN	
BACK BK	RUECKWAERTS RW	
BACKGROUND BG	HINTERGRUND HG	
BITAND	BITUND	(Commodore 64)
BITOR	BITODER	(Commodore 64)
BITXOR	BITXODER	(Commodore 64)
BLOAD	BL	(Commodore 64)
BSAVE	BBW	(Commodore 64)
BUTFIRST BF	OHNEERSTES OE	

BUTLAST BL	OHNELETZTES OL	
CATALOG	INHALT IH	
CHAR	ZEICHEN CHAR	(Apple II)
	ZEICHEN	(Commodore 64)
CLEARINPUT	LOESCHETASTE	
CLEARSCREEN CS	LOESCHEBILD LB	
CLEARTEXT	LOESCHETEXT	
CONTINUE CO	WEITER WT	
COS	COS	
COUNT	LAENGE	(Commodore 64)
CURSOR	BLINKER	
CURSORPOS	BO	(Commodore 64)
DEFINE	DEF	
DOS	DOS	
DOUBLECOLOR	DS	(Commodore 64)
DRAW	BILD	
DRAWSTATE	BILDZ	(Commodore 64)
EDIT ED	EDIT ED	
ELSE	SONST	
EMPTY?	LEER?	(Commodore 64)
END	ENDE	
ERASE ER	VERGISS VG	
ERASEFILE	VERGISSDATEI VD	
ERASEPICT	VERGISSBILD	
ERNAME	VERGISSNAME VN	(Apple II)
	VERGISS NAMEN VN	(Commodore 64)
FIRST	ERSTES ER	
FORWARD FD	VORWAERTS VW	
FPRINT		(Commodore 64)
FPUT	MITERSTEM ME	
FULLSCREEN	VOLLBILD	
GO	GEHE	
GOODBYE	ADE	
HEADING	KURS	
HIDETURTLE HT	VERSTECKIGEL VI	
HOME	MITTE	
IF	WENN	
IFFALSE IFF	WENNFALSCH WF	
IFTRUE IFT	WENNWAHR WW	
INTEGER	INT	
ITEM	ELEMENT EL	(Commodore 64)
JOYBUTTON	JB	(Commodore 64)
JOYSTICK	JS	(Commodore 64)
LAST	LETZTES LZ	
LEFT LT	LINKS LI	
LIGHTPEN	LGORT	(Commodore 64)
LIST	LISTE	

LIST?	LISTE?	
LOCAL	LOKAL	(Commodore 64)
LPUT	MITLETZTEM ML	
MAKE	SETZE MAKE	
MEMBER?	EL?	(Commodore 64)
NAMES	NAMEN	
NODRAW ND	LOESCHESCHIRM LS	(Apple II)
NOPRINTER	DRA	(Commodore 64)
NOT	NICHT	(Apple II)
	NICHT?	(Commodore 64)
NOTRACE	PROTOKOLLAUS PA	(Apple II)
	PA	(Commodore 64)
NOWRAP	RAND	
NUMBER?	ZAHL?	
OUTDEV	AUSGANG	(Apple II)
OUTPUT OP	RUECKGABE RG	
PADDLE	STEUER	
PADDLEBUTTON	KNOPF	
PAUSE	PAUSE	
PENCOLOR PC	FARBE	
PENDOWN PD	STIFTAB SA	
PENERASE	RADIERE	(Commodore 64)
PENUP PU	STIFTHOCH SH	
PRINT PR	DRUCKEZEILE DZ	
PRINT1	DRUCKE DR	
PRINTER	DRE	(Commodore 64)
PRINTOUT PO	ZEIGE ZG	
PRINTOUT TITLES POTS	ZEIGE TITEL ZT	
PROCEDURES	PROZEDUREN	
QUOTIENT	DIV	
RANDOM	ZUFALLSZAHL ZZ	(Apple II)
	ZZ	(Commodore 64)
RANDOMIZE	STARTEZUFALL SZ	(Apple II)
	SZ	(Commodore 64)
RC?	TASTE?	
READ	LADE	
READCHARACTER RC	TASTE	(Apple II)
	LT	(Commodore 64)
READPICT	LADEBILD	
REMAINDER	REST	
REPEAT	WIEDERHOLE WH	
REQUEST RQ	EINGABE EG	(Apple II)
	LIESLISTE LL	(Commodore 64)
RIGHT RT	RECHTS RE	
ROUND	RUNDE	
RUN	TUE	
SAVE	BEWAHRE BW	

SAVEPICT	BEWAHREBILD	
SENTENCE SE	SATZ	
SETDISK	ND	(Commodore 64)
SETHEADING SETH	AUFKURS AK	
SETSHAPE	FORM	(Commodore 64)
SETX	AUFX	
SETXY	AUFXY	
SETY	AUFY	
SHAPE	FEE	(Commodore 64)
SHOWTURTLE ST	ZEIGIGEL ZI	
SIN	SIN	
SINGLECOLOR	ES	(Commodore 64)
SPLITSCREEN	TEILBILD	
SQRT	QW	
STAMPCHAR	IGELZEICHEN	(Commodore 64)
STOP	RUECKKEHR RK	
TELL	AN	(Commodore 64)
TEST	PRUEFE	
TEXT PR	PRLISTE PL	(Apple II)
	PL	(Commodore 64)
TEXTBG	TEXTHG	(Commodore 64)
TEXTCOLOR	TEXTFARBE	(Commodore 64)
TEXTSCREEN	TEXTSCHIRM	(Commodore 64)
THEN	DANN	
THING	WERT	
THING?	NAME?	
TITLES	TITEL	
TO	LERNE PR TO	
TOPLEVEL	AUSSTIEG	
TOWARDS	RICHTUNG	
TRACE	PROTOKOLLEIN PE	(Apple II)
	PE	(Commodore 64)
TURTLESTATE TS	IGELZUSTAND IZ	
WHO	FORMZ	(Commodore 64)
WORD	WORT	
WORD?	WORT?	
WRAP	RANDSPRUNG RS	(Apple II)
	RS	(Commodore 64)
XCOR	XKO	
YCOR	YKO	

Weitere syntaktische Elemente sind die folgenden Sonderzeichen:

+ - * / = < > ( ) [ ] " ' : ;

Sie stimmen in der englischen und der deutschen Version überein.

Die obenstehenden Grundwörter wurden in diesem Buch mit sehr unterschiedlichem Gewicht behandelt; manche überhaupt nicht. Der Schwerpunkt wurde auf

diejenigen Sprachelemente gelegt, an denen sich die "Logo-Philosophie" (funktionales Programmieren, Modularität, Listenverarbeitung, Interaktivität, Erweiterbarkeit) besonders gut demonstrieren läßt. Alles was besonders geräteabhängig ist, und hierzu gehört (leider) auch der größte Teil der Graphik-Befehle, wurde dagegen in den Hintergrund gestellt. Wie schon in der Einleitung gesagt, soll dieses Buch nicht das Handbuch des jeweiligen Herstellers ersetzen.

Literaturhinweise

Abelson H.: Logo For the Apple II; BYTE / Mc Graw-Hill, Peterborough(New Hampshire) 1982
(deutsche Übersetzung und Bearbeitung durch H.Löthe, iwt verlag, Vaterstetten bei München 1983)

Abelson H. / diSessa A.: Turtle Geometry; The MIT Press, Cambridge (Massachusetts) 1982

Abelson H. / Sussman G.J. + J.: Structure and Interpretation of Computer Programs; The MIT Press, Cambridge (Massachusetts) 1985

Allen J.R.: Anatomy of LISP; Mc Graw Hill, New York 1978

Allen J.R. / Davis R.E. / Johnson J.F.: Thinking about [TLC] LOGO;
Holt, Rinehart and Winston, New York 1984

Böcker H.-D. / Fischer G.: Arbeitsmaterialien zum interaktiven Problemlösen mit Computerhilfe, Band 1 und 2; Hessisches Institut für Bildungsplanung und Schulentwicklung, Wiesbaden 1975

Brodie L.: Starting FORTH; Prentice-Hall, Englewood Cliffs (New Jersey) 1981

Dürr R. / Ziegenbalg J.: Dynamische Prozesse und ihre Mathematisierung durch Differenzengleichungen; Ferdinand Schöningh Verlag, Paderborn 1984

Engel A.: Elementarmathematik vom algorithmischen Standpunkt;
Ernst Klett Verlag, Stuttgart 1977

Goldschlager L. / Lister A.: Informatik: Eine moderne Einführung;
Carl Hanser Verlag, München 1984

Hermes H.: Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit; J.Springer Verlag, Berlin 1971 (2.Aufl.)

Hoppe U.: LOGO im Mathematikunterricht; iwt-Verlag 1984

Hoppe U. / Löthe H.: Problemlösen und Programmieren mit LOGO;
B.G. Teubner Verlag, Stuttgart 1984

Horn P. / Winston P.H.: LISP; Addison-Wesley Publishing Company, Reading (Massachusetts) 1981

Horowitz E.: Fundamentals of Programming Languages; Springer Verlag, Berlin 1983

Horowitz E. / Sahni S.: Fundamentals of Data Structures; Computer Science Press, Rockville (Maryland) 1976

Horowitz E. / Sahni S.: Fundamentals of Computer Algorithms; Computer Science Press, Rockville (Maryland) 1978

Knuth D.: The Art of Computer Programming
Vol. 1: Fundamental Algorithms
Vol. 2: Seminumerical Algorithms
Vol. 3: Sorting and Searching
Addison Wesley Publishing Company, Reading (Massachusetts) 1968/69

McDougall A. / Adams T. / Adams P.: Einstieg in Logo; Carl Hanser Verlag, München 1984

Noltemeier H.: Informatik I, II, III; Carl Hanser Verlag, München 1981, 1984, 1982

Papert S.: Mindstorms; Basic Books Inc., New York 1980

Schuppar B.: Logo Programmierkurs für Commodore 64 Logo und Terrapin Logo (Apple II); Friedrich Vieweg Verlag, Braunschweig 1985

Senftleben D.: Programmieren mit Logo; Vogel Verlag, Würzburg 1983

Tennent R.D.: Grundlagen der Programmiersprachen; Carl Hanser Verlag, München 1982

Thornburg D.: Discovering Apple Logo; Addison Wesley, Reading (Massachusetts) 1983

Wilensky R.: LISPcraft; W.W.Norton & Company, New York 1984

Wirth N.: Algorithmen und Datenstrukturen; B.G.Teubner Verlag, Stuttgart 1979

Worobjow N.N.: Die Fibonaccischen Zahlen; Deutscher Verlag der Wissenschaften, Berlin 1971

Stichwortverzeichnis

Das folgende Stichwortverzeichnis enthält auch die im Buche vorkommenden Prozedurnamen. Die Logo-Prozeduren sind stets mit Großbuchstaben geschrieben.

Jochen Ziegenbalg

Programmieren lernen mit LOGO

Einführung in das interaktive Programmieren mit zahlreichen Lernbeispielen

LOGO ist eine der mächtigsten Universalsprachen, die es heute für Microcomputer gibt. Dieses Buch bietet eine grundlegende Einführung in ein außerordentlich benutzerfreundliches, interaktives Programmiersystem mit vielen Beispielen, unter besonderer Berücksichtigung der deutschsprachigen LOGO-Versionen für den Apple II und den Commodore 64. Der Autor legt Wert auf fortschrittliche Konzepte der Informatik wie Modularität, Erweiterbarkeit, funktionales Programmieren, Listenverarbeitung usw.

Das Buch zeigt, daß LOGO bei weitem keine „Spielzeugsprache" ist (wie man wegen der so leichten Erlernbarkeit annehmen könnte). Es stellt auch dar, trotz des überdurchschnittlich reichhaltigen Befehlsrepertoires im Graphikbereich, daß LOGO eine ungewöhnlich effiziente, vielseitig wirksame und trotzdem leicht erlernbare Programmiersprache ist.



ISBN 3-446-14441-2